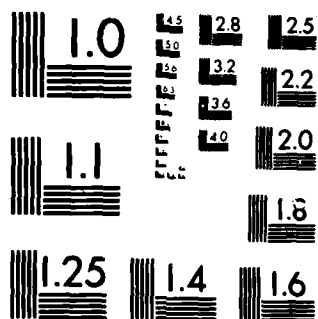


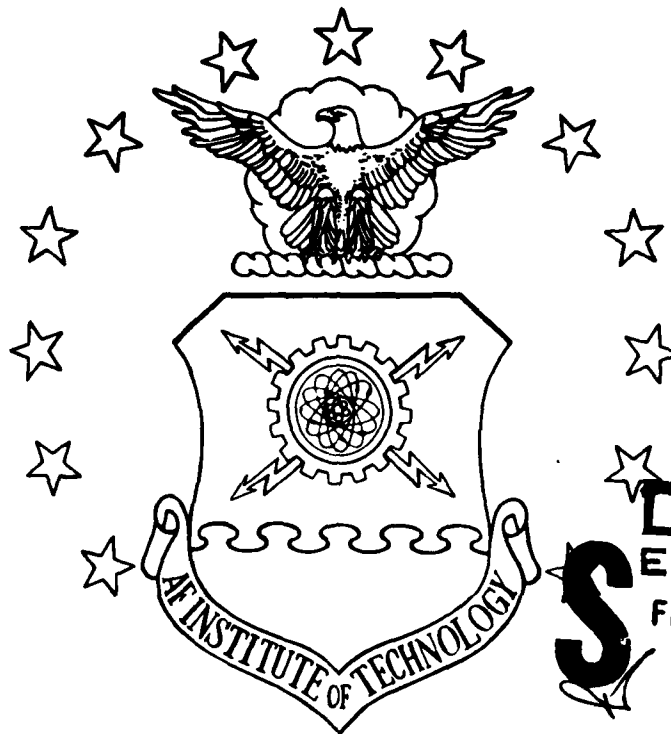
1/2

NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

AD-A163 951



DTIC
ELECTE
FEB 12 1986

S D D

**A CONTEXTUAL POSTPROCESSING EXPERT SYSTEM
FOR ENGLISH SENTENCE READING MACHINES**

THESIS

David V. Paciorkowski
Captain, USAF

AFIT/GE/ENG/85D-31

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

33 2 11 136

DTIC FILE COPY

(1)

DTIC
ELECTE
S D
D
FEB 12 1986

**A CONTEXTUAL POSTPROCESSING EXPERT SYSTEM
FOR ENGLISH SENTENCE READING MACHINES**

THESIS

**David V. Paciorkowski
Captain, USAF**

AFIT/GE/ENG/85D-31

DISTRIBUTION STATEMENT A

**Approved for public release;
Distribution Unlimited**

AFIT/GE/ENG/85D-31

**A CONTEXTUAL POSTPROCESSING EXPERT SYSTEM
FOR ENGLISH SENTENCE READING MACHINES**

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

DAVID V. PACIORKOWSKI, BSEE

Captain, USAF

December 1985

Approved for public release; distribution unlimited.

PREFACE

Under the current state of technology, most optical character recognizers (OCRs) could be described as machines that read with difficulty. A few OCRs are capable of accurate character recognition when the input text is highly constrained for image quality, typeface, and possibly, content. The purpose of this study was to design a postprocessor that could enhance the performance of OCRs which are used to read documents of a more realistic variety (e.g. no particular font or subject presumed, less than optimum image quality).

This postprocessor was developed using knowledge-based programming techniques and employing multiple knowledge sources concerning the structure and content of English words and sentences. The uniqueness of my approach was the emphasis on removing the constraints traditionally placed on the vocabulary and/or subject domain of the input text by other postprocessing systems. Although there are numerous other problems in OCR technology that must also be solved, this research is a very significant step toward the production of an accurate, and efficient automated reading machine.

I would like to thank my advisor, Dr. Mathew Kabrisky, for the encouragement and guidance that he provided, and for the autonomy of approach that he permitted of me. Most of all, I would like to express a sincere appreciation to my wife, Tammy, for her patience, her understanding, and her support given to me throughout this effort.

David Vincent Paciorkowski

Table of Contents

	Page
Preface	ii
List of Figures	v
Abstract	vi
I. Introduction	1
Background	1
Problem Statement	6
Approach and Scope	7
Assumptions	9
II. General Solution Design	12
Resolving Uncertainty with Expert Systems	12
Hierarchical Processing	17
Lower Level Knowledge Sources	20
Higher Level Knowledge Sources	24
Implementation Programming Language	28
III. Specific Design and Implementation	30
Controlling Modules	30
Thresholding and Word Hypothesization	35
Spelling Expert Subsystem	37
Supplementary Knowledge Sources	44
Other Word Level Knowledge Sources	48
Syntactic Expert	51
IV. Testing and Results	53
Review of Goals	53
Testing Method	54
Observations	55
V. Summary, Conclusions, and Recommendations	69
Summary	69
Conclusions	70
Recommendations	71

	Page
Appendix A: Postprocessor Program Listing and Data Base	73
Appendix B: Fourier Distance Matrix for Simple Character Set . .	108
Appendix C: Test Set of Simple Characters	114
Appendix D: 2D-FFT Program Listing	117
Bibliography	128
Vita	130

Accession For	
NTIS CRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

List of Figures

Figure	Page
1. Control Module Hierarchy	34
2. Letter Sequence Processing Flow	42
3. Example of Postprocessing Improvement	57
4. Example of Postprocessing Improvement	58

ABSTRACT

Knowledge-based programming techniques are used in an expert system to reduce uncertainty for optical character recognition by combining evidence from several diverse knowledge sources cooperating in an hierarchical configuration. The postprocessor development focused on a system for generic text input that is not constrained to a fixed vocabulary or particular subject domain. The key element to the system's effectiveness is a spell checking algorithm that is not operationally bounded by an enumerated lexicon or biased by statistical sampling. The postprocessor system is also design to interface almost any type of OCR front-end methodology.

Expend: Texas

image processing

A CONTEXTUAL POSTPROCESSING EXPERT SYSTEM FOR ENGLISH SENTENCE READING MACHINES

I. Introduction

Background.

The increasing use of automatic data processing and the widespread use of computers to solve complex problems have compelled the importance of evolving efficient means to enter information into these machines. At present, almost all textual data to be used by computers is initially translated into a machine recognizable form by typing on a keyboard. An alternative interface to these computers, capable of accepting input by human speech or by optically scanning printed documents, would ease the process of initial data entry and increase the overall efficiency of automated information processing.

One of the interface alternatives currently being researched is the optical character reader (OCR). The OCR is an image processing device used to identify letters, other typed characters, and in some instances hand-written text. The image processing performed by an OCR can be generalized into three basic steps:

1. Digitization - conversion of the optical image into a discrete mapping of picture elements (pixels), each quantized on a grey scale to represent signal intensity.

2. Preprocessing - locating individual characters or character features; may also include image enhancement such as conversion of the digitized image into binary grey scale format to facilitate the separation of text components such as lines, words, and characters.
3. Identification - picking the best match of the character image to a known character; this may be accomplished by finding the best alignment between the image and a character template, or by conversion of a set of feature measurements of an individual character into an n-dimensional vector, where n is the number of individual features describing the composition of the character, and matching the vector representing each optically scanned character with the closest prototype vector that represents a known character.

There are numerous optical character readers currently being produced for specialized purposes such as bank check reading, postal zip code reading, and even some page readers for text entry. But reading machine technology is still in the early stages of development. Those machines which are commercially available at present perform under significant restrictions or with limited accuracy. One of the most limiting restrictions for current machines is their dependence on specific letter spacing, specific letter sizes, and specific letter fonts for the printed material used as input. As long as the letters in the text to be read are well separated, in known positions, and belong to a particular type set, an OCR employing basic template matching techniques can successfully read the material. However, printed

material comes in countless variations of size and font. The spacing between letters is also often not uniform or even nonexistent. A reading machine which could operate accurately without the strict restrictions that current reading systems place on printing style and format would have widespread applications and would represent a significant improvement in reading machine technology.

The increasing merger between the pattern recognition research and artificial intelligence techniques has opened new avenues toward the development of accurate, automated, multifont reading systems. Character recognition by simple template matching cannot be successful in overcoming the obstacles of multifont reading. The computational resources needed to store and to search through all typeset variations of all characters would make a template system impractical to produce and use, if not impossible to develop. Instead, the features used to identify characters must be chosen to form intrinsic definitions of each character. Besides some of the new approaches to improve the identification abilities of optical character readers with unique feature extraction, knowledge-based programming and mathematical algorithms are being applied as postprocessors to the image recognition process to improve the overall performance of text reading systems.

A system considered to be state-of-the-art for multifont reading, the Kurzweil 4000 Intelligent Scanning System, uses the general shapes of letter components such as loops, horizontal lines, and concavity to recognize characters. The use of shape features by this system was chosen to obtain multifont reading capabilities and the ability to read proportionally spaced type. The concept behind this design choice is

that each letter maintains a basic form, or one of a few basic forms, for all sizes and styles of print found in text (1:111).

Promising results in pattern recognition research, conducted at AFIT, for the development of a general text reading machine makes use of two-dimensional discrete Fourier transforms (2D-DFT) in the identification stage of optical character recognition. This technique has been successful in identifying isolated letters and isolated whole words with considerable independence toward the size print and toward the font used in the text (2; 3; 4). The 2D-DFT was proposed by Kabrisky as a mathematical model of the human visual information processing system, to suggest how optical images are represented and processed in the brain. Subsequent research has given increased credibility that the low frequency spatially filtered 2D-DFT is a Gestalt representation of an image. Therefore, the low frequency filtered 2D-DFT of a character image represents the essence of the character's identity independent of its specific form. A letter "B" has "B-ness" regardless of whether it was printed in a plain gothic font or in a highly decorative font with swashes and seriffs (2:2; 3:4-5; 4).

The intrinsic letter shapes used by the Kurzweil 4000 and the use of the low frequency 2D-DFT as a Gestalt representation are two examples of how choosing the proper descriptive features can increase reading machine capabilities toward efficient automated multifont text processing. However, in the application of any character identification technique to printed pages of text, several problems are anticipated to degrade recognition performance. Lack of control over the quality of the print is one aspect of the reading environment which will create

uncertainty in the letter identification process. All letters may not be well formed and of uniform contrast. Spaces in the ink, or variations from too much or too little ink for any particular letter could cause a misinterpretation by the reading machine. The lack of separation of letters within a printed word is another factor which reduces the reliability of the automatic reading process. Parts of adjoining letters could be included in the optical processing of each letter. This can often cause a letter to resemble other letters. One other important source of uncertainty for the reading process is noise. The noise introduced by the image detection and signal digitization equipments will also increase the likelihood of character recognition errors. The OCR using a multifont character identification process is not in itself a sufficient solution to the automated text reading problem. Some enhancement is necessary to improve its reliability and increase its speed of independent operation.

There is a lot of information contained in the structure and content of English text which is not being used by OCR systems to improve the probability of correct character recognition. English sentences conform to a large but well-defined set of constructual arrangements based upon the part of speech which each word fulfills. There is also much information which can be derived from the meaning of each sentence that could be used to resolve uncertainties about individual letters or words contained in a sentence. The sequences of letters which form words in any language are constrained by a limited inventory of allowable sound sequences for that language.

Consider the example of a person trying to read a document which was copied on a poorly maintained photocopier. Anyone who has had such an opportunity has found that the text can be read successfully even when the print is very distorted and significant portions of many of the letters are missing. Under such circumstances the human reading mechanism uses a complex and unknown process to complete the task. The natural redundancy of information in the English language must help significantly. But, the reader is also using his experience and understanding of the language to solve the problem. The reader must generate hypotheses about each of the uncertain letters and words. Eventually, one of the hypothesized sentences which is composed of recognizable words, is grammatically complete, and is consistent in content with the content of other sentences read is chosen for the solution. A human reader attempting to read an unknown, foreign language under these same circumstance would not be as successful as a reader in his native language. The performance difference comes from the understanding of the word structures, the word meanings, and the entire concept presented by the sentences.

Problem Statement.

One logical approach to improving the performance of OCRs is to imitate the human reading process. An OCR attempting to read letters which appear incomplete or distorted to its recognition process is similar to the human reader attempting to read text from a poor quality

copy. Knowledge of the language and information contained in readable portions of the document must be used together to solve the problem. Expert systems are computer programs which adapt knowledge sources into rule-based processes for the purpose of solving difficult problems. It should be possible to improve the performance and reliability of an OCR by combining it with an expert system for contextual postprocessing. The expert system would make use of apriori knowledge about legitimate letter sequences in words and would process the syntactic and semantic information available in English sentences. This thesis research will explore the use of semantic, syntactic, and other word and sentence knowledge sources in an expert system that will combine the evidence available from multiple sources to reliably identify characters in printed English text.

Approach and Scope.

The focus of this research is to devise a contextual postprocessor that may be used to improve the accuracy and throughput of text reading systems in general. The various levels of expert systems to be used by the postprocessor will be chosen so that they do not restrict the image processing methodology of the OCR front-end, and so that the textual data entry process can be performed with efficient automation. Automation efficiency will be provided through increased recognition accuracy and a reduced need for operator intervention for machine training and correction.

The contextual postprocessing system developed for this thesis will consist of a hierarchical control structure which will interface with an optical character reader and several component expert systems. The design of the system will be such that almost any type of OCR could be used with this system, including a template matching OCR. The only requirement of the OCR is that its character identification methodology allow it to produce a weighted list of choices for each character being identified. For testing purposes, the OCR front-end to this system will use the 2D-DFT method of character identification. The input from the OCR will be offline or simulated.

The expert systems used by the postprocessor will be arranged to provide additional evidence concerning the likelihood of each character choice as characters are combined to form various structures from letter sequences (n-grams) through sentences. The evidence provided by the various knowledge sources will be used to either eliminate character, word, and sentence hypotheses or to endorse the plausibility of those hypotheses. The association of "probability" measures with competing letter, word, and sentence hypotheses will be used by the control structure to combine the evidence provided by the sources of information and eventually establish a best choice decision at the sentence level. Some information will be extracted from the sentences that have completed processing and this information will be used toward the processing of the sentences which follow within the same body of text.

The key expert system in the processing hierarchy will be a low level but powerful knowledge source constructed to provide information about spelling rules for valid syllables and letter sequences in the

English language. This knowledge source will be used to reduce the set of solution hypotheses to be processed through the other knowledge sources by eliminating impossible words. Another expert system to be used by the postprocessor will be a syntactic parser. This knowledge source will apply rules of grammar to eliminate non-conforming sentences. Research for sentence parsing is well established; therefore this thesis will not attempt to develop a parser unique to this system. The parser interface will be simulated so that future research using this postprocessing system can be modularly adapted to whatever parser is available.

Some use of semantic information will be incorporated into the contextual postprocessing system; however, complete semantic analysis at the phrase level, sentence level, or higher is not practical. Existing approaches for text understanding systems are only capable of operating within specific subject domains and with very restricted vocabularies. Semantic influence in the system will be restricted to a process resembling the short term memory function employed in the speech recognition system, SPEREXSYS (5).

Assumptions.

The reading machine process developed by this research project will not operate free of all constraints. It is assumed that the text to be used as input is composed of grammatically correct sentences. All words should be in the English language or if the word is newly coined it

should conform to typical English phonemic structure. The number of different English words which may be used in the input text is not constrained because the postprocessing system will not depend upon an apriori listing of vocabulary. The spelling of words should be reasonably correct. Incorrect spellings may be identified as unrecognizable words, or in a few cases corrected providing there is sufficient supporting evidence.

Another important consideration is the accuracy of the OCR front-end to this system. If the OCR were completely accurate, there would be no need for this system. However, the recognition accuracy of the OCR front-end will impact on output accuracy of the combined reading system. Because the postprocessor is attempting to use knowledge derived from the structure and content of the text, some portion of the text must be recognized accurately. Therefore, a recognition threshold for the OCR will be use to allow some of the characters to be considered as correctly recognized without postprocessing; treating all input characters with uncertainty would be computationally prohibitive toward real time processing. The postprocessor will have more information on which to base its decisions, and thus produce more accurate decisions, if the accuracy of the OCR allows more characters to be interpreted as correctly recognized without postprocessing. This postprocessor is expected to improve the performance of almost any OCR. However, the postprocessor will only be considering the top few choices presented by the OCR for each character position. In order to achieve a performance improvement, it is most important that the correct choice for the character being read is in the top few choices of the OCR's output. As

the number of characters which can be considered correctly recognized without postprocessing decreases and the ability of the OCR to provide the correct letter in the top few choice decreases, the improvement provided by the postprocessor will eventually decrease to a negligible amount. This postprocessor is intended for, although not limited to, those OCRs which are performing just a little bit shy of the reliability needed for efficient automation of text processing.

II. General Solution Design

Resolving Uncertainty with Expert Systems.

As mentioned earlier, an expert system is a knowledge-based computer program designed to give advice or solve difficult problems concerning a specialized subject. Expert systems have been developed for many applications which include medical diagnosis, analysis of chemical and geological data, engineering design, and speech understanding. The basic components of such a system are a database of information, a set of inference rules to reason with, and a control strategy for applying the rules to the database and for resolving conflicts in the reasoning process. The simplest form of these systems uses well-defined rules of mathematical deduction or predicate calculus to draw conclusions from consistent statements in the knowledge base. But there are many applications where the data is not readily manipulated by the deterministic rules of predicate logic.

Alternate techniques must be used when the data may be considered unreliable. Situations producing unreliable data include those when the evidence is questionable or unavailable, those when data descriptions include relative measures that are not precisely quantifiable, those when the conclusions that can be proposed are not absolute, or those when assumptions must be made based upon likeliness or lack of contrary evidence. Character recognition fits into several of these categories.

The problems of print discontinuity and extraneous markings in the input text are two obvious examples of questionable evidence being used in the character recognition process. The continuing efforts to find an accurate multifont feature set to be used for character recognition demonstrate that the image data is not precisely quantifiable with respect to the problem. The use of thresholds by many OCRs for a recognition/non-recognition decision in the identification process shows that the processing conclusions are often non-absolute. Some of the general approaches used by expert systems to cope with uncertainty or unreliability in the data are nonmonotonic logic, probabilistic modeling, evidential reasoning, and fuzzy logic (6:175; 7:93-96).

An expert system employing nonmonotonic logic allows hypothetical statements to be added and deleted from the database in response to the admission of new knowledge. This is accomplished through such methods known as default reasoning and dependency-directed backtracking. Default reasoning makes assumptions about relevant conditions in the absence of any contradictory information; the most probable choice (default value) is assumed if no data is provided. These default assumptions and all further inferences contingent on the default assumptions are subject to revision as new evidence is supplied to the system (6:176; 7:73-75). When a new element of information reveals an inconsistency, assumption and inference statements which were added to the database must be traced back to the original source of error and withdrawn. A chronological backtracking of the reasoning process would be inefficient and wasteful because assumptions and inferences not dependent on the assumption in error would be erased from memory during

the backtrack and probably regenerated in the near future. By keeping track of the supporting assumptions and inference rules which justify each statement added to the database, dependency-directed backtracking withdraws only the assumptions now believed to be in error and any inferences derived from those assumptions (6:179; 7:75).

Nonmonotonic reasoning systems are useful in several circumstances. When the lack of complete information would stop the problem solving process, default reasoning maintains the momentum. When the problem solution requires the generation of temporary assumptions and partial solutions, dependency-directed backtracking provides an efficient means of revising small components of the total solution. When the information is provided to the database over an extended period of time or when the information in the database changes states over time, a nonmonotonic system is an effective method of maintaining current theory for a problem solution (6:179). However, the nonmonotonic reasoning approach does not suit the postprocessing application. Although this method may be capable of improving the accuracy of character reading machines, the amount of backtracking computation involved would not be efficient in automating the reading process. This method of continuous second guessing does not fit well into the human model of character recognition.

Fuzzy logic is based upon a specialized mathematical set theory and is appropriate for interpreting imprecise quantifications of data (7:95). In fuzzy logic, descriptive modifiers such as large, small, and many are characterized by specific ranges of values, each associated with a probabilistic measure (e.g. the value is between 10 and 1,000

with a possibility of 15%). The use of fuzzy logic in expert systems has not been as widespread as some of the other reasoning methods. This is largely due to the complexity of mapping the mathematical theory into the domain of expert interpretations for specialized problems (8).

Probabilistic modeling is the application of statistical data in quantifying the reliability of input data and the likelihood of the conclusions. In systems which process high volumes of sensor data, statistical calculations such as standard deviation and correlation are used to characterize the data. When data from multiple sources are combined, joint probability functions are used to calculate the conditional probability of a concluded event (9). Probabilistic modeling is a mathematically supported method of dealing rigorously with data uncertainty. However, there are many drawbacks to its widespread application. The large amount of data needed to determine the probability functions is not always available. The large number of interactions between observations is often so difficult to understand that conditional independence of sources is assumed without thorough justification. The constraint that the sum of the probabilities of possible events must equal one, makes it difficult to modify the knowledge base and rule set. The accuracy of the probabilistic conclusions is dependent upon the set of hypothetical outcomes being complete and mathematically disjoint (6:192-193; 7:94). Of these four major approaches being used in expert systems to reason toward a solution in the presence of unreliable or inconsistent input data, probabilistic modeling seems to be the approach used most often in contextual postprocessing research.

Probabilistic postprocessors use statistical data concerning individual letter occurrence frequencies or letter sequence (n-gram) frequencies to estimate the correctness of the text interpreted by the OCR (10, 11). This approach has had some success but the success was dependent upon a close match between the source of statistical data and the text being processed. Shannon, who was among the first people to recognize the dependencies between letters in natural languages, suggested that parameters involved with predicting strings of English text are dependent upon the particular text involved (12). A variation on the probabilistic approach which combines subjective estimates of probabilities and a means of combining positive and negative support for a proposition is called evidential reasoning.

Evidential reasoning uses a confidence scale ranging from +1 to -1, distinguishing between supporting and refuting information (13:206; 14). Conclusions are assigned a certainty value that results from the product of the probabilistic rating of the information used and the certainty factor associated with the inference rule used. The strong point of this methodology comes from a complicated, but well-formed set of rules for combining weighted evidence from multiple sources. These rules are referred to as the Dempster-Shafer theory (14; 15). Basically, the combination of two observations results in a measure of belief that is equal to the measure of belief associated with one of the observations plus an incremental portion of the belief associated with the second observation. Thus, several items of information with relatively small individual probabilistic weightings can combine to increase the confidence value of an hypothesis (6:194).

Many of the more successful expert systems, such as MYCIN and PROSPECTOR, are using some variation of evidential reasoning to integrate the knowledge contributions of many diverse information sources. In these systems some of the information sources used to produce a conclusion are naturally quantifiable such as sensor data and the results of diagnostic tests. But other sources of information are simply the subjective opinions of specialists biased by their experiences. The key advantage of evidential reasoning is that it is a formal means of combining knowledge from a wide range of sources. The model for the human reading process, as introduced in the first chapter, uses a wide range of knowledge sources in producing a solution. An expert system designed to imitate that model by combining information and judgements about text hypothesis at various levels from the image data of single characters to the grammar conformity of complete sentences will require the flexibility of evidential reasoning approach.

Hierarchical Processing.

Many expert systems use several different sources of information, especially syntactic and semantic knowledge, to select a confident interpretation of optical or speech signals. These systems typically conform to one of two basic architectures. One architecture is the modular approach in which discrete knowledge sources are arranged in an hierarchical order and are scheduled to process the signal data independently. The other architecture is the compiled knowledge

approach in which all the knowledge about the particular problem domain is integrated into a large network and processing is similar to searching for an optimum path through the network (16:332-342).

The HEARSAY speech understanding systems are the most frequently used examples of the modular approach to using multiple knowledge sources. The HEARSAY-I system used three separate knowledge sources: an acoustic/phonetic source which included information about variance between speakers and about environmental noise, a syntax knowledge source, and a semantic knowledge source (16:343-344). Although the knowledge sources were very domain dependent, each acted as independent processors and could be substituted for or modified without effecting the operation of the other modules. This is the key advantage of the modular approach. The ability to add, delete, modify, and replace knowledge sources within the expert system is extremely useful in the research environment. The HEARSAY-II system differed from its predecessor in a few ways. HEARSAY-II operated in a more complex problem domain and used more knowledge sources that had smaller areas of specialization. But the key difference was in the overall control of processing. The HEARSAY-II system made more efficient use of its computational resources by restricting the processing of hypotheses by the various knowledge sources to a limited amount of best choices. This was primarily accomplished by ordering the lower level knowledge sources into an hierarchy where processing at one level must be complete before processing at the next higher level is started (16:343-348).

In contrast to the HEARSAY systems, the HARPY speech understanding system used the compiled network approach. The HARPY system was more

efficient and accurate than the HEARSAY systems when compared in the same problem domains(16:351). The improvement came from two basic concepts. The complete set of possible solutions is assembled in the system network. Therefore, the set of competing solution hypotheses is smaller. This system had better accuracy because it had fewer incorrect solutions competing with the true solution. In the modular system, it is possible that the true solution is never assembled into an hypothesis. The other advantage of the network system is speed. Efficient search heuristics which restrict backtracking can be used on the precompiled network to produce a solution more quickly than if search path needed to be generated as the search progressed.

But the compiled network approach has inherent disadvantages that are important considerations in this research project. The obvious disadvantage is that whenever a knowledge source is changed, the entire network must be reconstructed (16:337). This is a time consuming and complicated task because the changes must be incorporated explicitly at an affected branch of the network. The other important disadvantage is that in order to explicitly define the network, the problem domain must be quite constrained (16:337). This requirement may be acceptable when working in domains with limited lexicons and syntax such as the postal address reader, but it is not acceptable when the problem domain is a general text reader where the full range of sentence syntax and an unrestricted vocabulary comprises the possible input. In order to concentrate on the representation and cooperation of diverse knowledge sources in this research system, the modular architecture with hierarchical processing was used.

Lower Level Knowledge Sources.

The knowledge sources/experts described in this thesis as lower level sources are those which perform postprocessing of text constructs that are whole words or smaller. The most common approaches used in research of postprocessing systems operating at this level are the use of statistical estimation concerning possible letter sequences, and the selection or confirmation of hypothesized words based on an enumerated lexicon. Both methods have been successful in improving character recognition for texts within limited domains.

The statistical estimation approach recognizes that there are dependencies among characters as they appear in words in a natural language. The premise behind this approach is that natural languages can be approximated by a Markov source (10; 12:536). Letter transition frequencies, gathered from sample texts, are used to determine the most probable letter choice in accordance with the frequencies that various letters follow the previous letters. In some systems only one previous letter is considered; these systems assume a second-order Markov process for their approximations of digram probabilities. Improved error rates are achieved by using statistical data about trigrams, three-letter combinations, to choose the most probable letter. The third-order Markov approximation has a high resemblance to ordinary English text (10). Using fourth-order or higher order approximations require significant increases in computational and data storage resources which are

difficult to justify by the anticipated performance improvements. A third-order system has 2^3 or 17,576 trigram statistics to store and process while a fourth-order system would require 2^4 or 456,976 statistics about 4-grams.

The statistical approach has a key advantage over the lexical approach. By strictly relying upon letter transitional properties at the n-gram level, the system is not constrained to an enumerated vocabulary. The vocabulary is constrained by the source of the n-gram statistics. Some legitimate words may not be accepted by these systems because an unusual letter combination may not have been present in the source texts used to generate the letter transition probabilities. The success of the statistical estimation approach is dependent upon the source of the estimation statistics. When the source of the estimation statistics match the contextual properties of the material being recognized, the performance is better than if the estimation statistics were derived from a general source and applied toward recognition of text with very unique characteristics (10:550). For example, the statistics derived from typical narrative text would be different from the statistics derived from a list of names; therefore, the letter transition predictions made with both sets of source statistics would be different. One statistical set will have a higher percentage of correct predictions. Obviously, the set which most closely resembles the text being processed will perform better. An interesting side effect of the statistical approach is that in some cases this process has been able to correct spelling errors that had been present in the original text being recognized (11).

The lexical approach to contextual postprocessing is principally a dictionary lookup method. In simple lexical systems the word hypothesis is searched for in the dictionary until a match is found. If no match is found the word is a reject error, flagged as non-recognizable by the system. More sophisticated systems will not immediately reject a word that is not found in its dictionary. Instead, the system may change some of the letters in the word to find a match. This type of system will have a much smaller reject error rate, but the amount of substitution errors will increase. Many of the letter substitution lexical systems use digram probabilities to choose the substituted letters. However, this use of letter transition statistics does not expand the system vocabulary; the entire lexicon accepted by the system must still be contained in its dictionary.

In limited problem domains, where the entire set of possible words can be represented in the dictionary, lexical postprocessing is very effective. A very large vocabulary is required to apply the lexical approach to the general text reading problem. As the dictionary size grows, the amount of storage required and the amount of computation required increases. Also, it is more likely that incorrect words will be substituted for words not in the dictionary because the dictionary contains more words closely resembling the actual word. It is impractical to have the dictionary contain all possible words; therefore, any lexical system is subject to having a high reject error rate for some texts not typified by its vocabulary (i.e. text containing specialized medical terminology). High non-recognition rates are not desirable when one of the key interests of the system design is to

increase automation. Reject errors require a lot of operator assistance during the recognition process or a lot of proofreading after the recognition process is complete.

Although there are lexical based knowledge sources used in the expert system designed for this research, those knowledge sources are not the primary processors in the system. The key low level processor in this system was designed to remove the constraints of a limited vocabulary without containing the bias of statistical data. This processor is similar to those used in the statistical approach systems because it evaluates words at the n-gram level and it is not restricted to a enumerated vocabulary. But the important difference from the statistical processors is that this knowledge source is not biased by the source of statistical data because this sub-system only provides a determination of letter string legitimacy and leaves the ranking of choices to the sensor data information and other knowledge sources. The key low level processor uses variable length n-grams and word position dependent combination rules to determine which hypothesized words conform to normal English spelling patterns.

There are several other knowledge sources operating at the lower level. Some of these are used to rank the word and letter choices, while others pass judgement on the permissability of word hypotheses but using different rules than those used by the key processor. For instance, one processor knows rules about the positioning of apostrophes within words; those words which do not conform are eliminated from further processing. Not every possible low level knowledge source was implemented; only a few simple ones were used to demonstrate the types

of information which could be brought to bare on the problem. Additional knowledge sources could be easily added to the modular design if this methodology is used again in future research.

Higher Level Knowledge Sources.

The knowledge sources/experts described in this thesis as higher level sources are those which perform postprocessing of text constructs that are larger than individual words. Although it is possible to have these knowledge sources operate with phrases and partial sentences, it is easier to maintain a black box approach to employing higher level experts if this level of postprocessing focuses on complete sentences. The general purpose of including the higher level knowledge sources is to add constraints to the possible combinations of words that can be hypothesized by the lower level experts. Two areas of language knowledge that can be applied to sentence level hypotheses are syntax and semantics.

Syntactic analysis examines the word arrangement within a sentence to determine if and how the sentence conforms to the rules of grammar. This analytic process is called parsing. Parsing determines the relationships of each word within a sentence to the other words in the sentence. Sequences of words which violate the rules of grammar are rejected by a parser. There are many different approaches to parsing English sentences. These different approaches to parsing represent attempts to improve efficiency and to produce better interpretations for

sentences which conform to more than one grammatical structure. Many of the different parsing schemes are analogous to the differences in search techniques; some parsers use extensive backtracking, some parsers search for all syntactically correct solutions, and some parsers use heuristic directed search to find only one syntactic solution. There are also some parsers which rely on semantic information to find the correct interpretation of a sequence of words. Parsers using semantic grammars usually examine only the keywords of a sentence; therefore, they would not be appropriate for resolving ambiguities which can occur with any word in a sentence.

Semantic analysis is very complicated and is not easy to define. One simple definition of semantic analysis is determining the mapping between the syntactic structure of a sentence and some appropriate relationship between the objects represented by the words in the sentence. Very simplistically stated, semantic analysis interprets what the sentence means and if that meaning makes sense.

Currently, the basic approach to semantic analysis is to match the sentence keywords to either a frame or script knowledge representation and then assign other words in the sentence to the various slots in the frame or script. The frame and script systems have expectancies about which words can be associated with the keywords identifying the frame or script. For example, if the keyword for a particular frame was the verb "drink," then the subject of the sentence would be expected to be a person or animal, and the object of the sentence would be expected to be a liquid. Scripts form similar expectancies, but the scope of those expectancies may extend over several sentences or the entire text.

The basic properties of parsers and semantic analyzers suggest that they are both operationally constrained to limited domains. The analysis of syntax presupposes knowledge of the parts of speech which each word can fulfill. Therefore, parsers are constrained to operation within a fixed vocabulary. Semantic analyzers are also constrained to operation within a fixed vocabulary. Semantic analyzers require knowledge of how each word can be used as parts of speech, how each word belongs to classes of objects and actions, and how each word forms expectancies toward other classes of actions and objects. The complexity and size of the knowledge base required for semantic analysis increases nonlinearly as the subject domain (vocabulary) increases. It would not be feasible to use the current methods of semantic analysis in the postprocessing expert system for reading generic text.

However, the use of syntactic analysis may be adaptable to the generic text postprocessor. If only one or two words in an occasional sentence are not contained in the vocabulary for the parser, then it may be possible to hypothesize the part of speech which the word must fulfill and store this hypothesis for use or confirmation while processing the remaining sentences in the text. If words not in the parser vocabulary are used more than once in the text, then this method could increase the benefits of parsing while postprocessing texts that are not restricted to predetermined vocabularies. There are many circumstances which increase the likelihood that an unusual word will be repeated in a text. The word which is unfamiliar to the parser vocabulary may be a word of preference in the author's vocabulary. Also, if the unfamiliar word is related to the specific topic of the

text, then the word is likely to appear again. Similarly, words are repeated in written text to form transitions between the thoughts expressed in one sentence to those expressed in following sentences.

The requirement that the parser be able to hypothesize the part of speech for an unknown word makes it difficult to use existing parser systems for this postprocessor. However, since the parser information will not be used in semantic processing, the parser does not need to determine the optimal grammatical interpretation of the sentence. The parser is only required to determine if a correct grammatical interpretation exists. This may help in the modification of existing parsers to fit the needs of this system. Parser operation was simulated in this thesis because there was not sufficient time in the thesis schedule to adapt or develop a parser which would provide the operations described above.

Noticing the phenomenon that words tend to recur throughout a body of text suggested that there is an important source of information which has not been used by any of the lower level or higher level experts mentioned so far. If the words used in each of the accepted sentence hypotheses were remembered, these words could beneficially influence the selection of word hypotheses at the lower processing level. This is somewhat similar to the implementation of a short term memory as used in the speech recognition system, SPEREXSYS (5). This knowledge source should produce the effect of having a domain specific lexicon from which word hypotheses can be selected with increased confidence.

The benefits of a short term memory knowledge source would not be as significant at the beginning of text being processed as in the middle

or end of that text. It may be possible to offset the learning curve of this knowledge source. Some words have a tendency for recurring in many types of texts while other words have a tendency for recurring in texts which are subject domain specific. The short term memory does not need to start off blank at the beginning of each new text. Instead, this knowledge source could contain a short list of words which have a tendency to recur in a wide variety of texts, such as the most likely words compiled from the Brown Corpus (4). This way the benefits of this knowledge source could be more consistent for all parts of the text being processed.

Implementation Programming Language.

Several programming languages were considered for implementation of the expert system proposed in this thesis. At first, Fortran was considered because reading machine research previously conducted at AFIT was accomplished in Fortran. Fortran would be a good selection if this system was to interface with the existing programs for character recognition. But the design of this system was intended to be independent of the OCR front end. The OCR front end could be simulated and Fortran was not necessary to simulate the OCR output.

Lisp and the available expert system building tools, such as KEE, were considered. These programming environments are well suited for implementing expert systems. However, a Lisp based program would probably not be easily transproted to a system supporting the OCR

front-end processing. Optionally, the OCR program could be moved to the Lisp environment, but the vast amount of mathematical computation used in those programs would not adapt well. Also, the expert system building tools were not used because I did not feel I had enough time to learn the tools during my thesis schedule, and because an attractive alternative was available.

Pascal was chosen for the implementation language for several reasons. Pascal is well suited to modular design. The program code written in Pascal is very similar to psuedo code used to design algorithms. The resemblance to psuedo code makes it easier to use the ideas explored by this thesis in future research in almost any other programming environment and to transport this design to whatever system is driving the OCR front-end. Another important factor is the mathematical processing used to weight and combine evidence seemed easier to accomplish in Pascal than in a Lisp environment. And finally, I was already familiar with programming in Pascal. The postprocessor prototype was implemented on a Vax computer using Berkley Pascal. Portions of the prototype were developed using Turbo Pascal on a minicomputer.

III. Specific Design and Implementation

Controlling Modules.

The control modules for the postprocessing expert system are arranged in an hierarchical fashion. This arrangement provides a flexible framework for experimenting with the textual constraints produced by the individual and cooperative efforts of diverse knowledge sources. However, in a few instances the control elements are not isolated from the knowledge sources. This occurs at the lower level where some knowledge sources are embedded with control statements for other knowledge sources. Control statements were grouped into modules as much as possible, but there were occasions when the optimum time to activate the processing of a supplementary knowledge source was during the processing of the primary knowledge source.

The expert system contains five hierarchically arranged control modules, which includes the main program module. Each of these modules is responsible for using utilities to manipulate text structures for the next phase of processing, triggering the knowledge subsystem processors that are within its scope of control, and passing control at appropriate times to the control block which is the next lowest in the control hierarchy. Control returns upward through the hierarchy only after all actions associated with a particular text construct are completed. As control returns to the higher modules, flags are also passed to indicate

the status of the portion of text being processed (e.g. word completed, sentence completed). All of the control modules reflect the modular approach of the expert system by containing a relatively small amount of code with distinct calls to individual utilities and expert subsystems.

The main program module is responsible for starting and stopping the postprocessor. It also makes procedural calls to an initialization utility to load the data bases for the various knowledge sources, and to output utilities to store completed sentences and to print a copy of the text which resulted from the completed postprocessing effort. The main module operates the postprocessor by continuously requesting the next control module, MakeSentence, to provide processed sentences. Processing is stopped when a control flag indicates that there is no more text to be processed.

The MakeSentence module is the primary controller for the higher level knowledge sources. It passes control down the hierarchy by requesting the module, GroupWords, to provide the hypotheses for the next sentence to be processed. Once the hypotheses are received from GroupWords, the MakeSentence module processes them with the higher level knowledge sources. Since unrestricted domain semantic analysis is not feasible within current technology, only syntax is used to determine which of the sentence hypotheses are likely to be correct. Selection of the best sentence hypothesis is done by a utility under the control of the MakeSentence module. After selecting one hypothesis, MakeSentence invokes another knowledge source to extract new words from the sentence so that additional weighting can be given to future hypotheses which reflect the repetition of words used in previous sentences.

GroupWords is considered as one of the control modules for the higher level text processing. Although it does not invoke any knowledge source, it is responsible for the interface between the lower level and higher level knowledge sources. GroupWords forms the output from the lower level processing into sentence hypotheses which can be processed by the higher level knowledge sources. If the higher level knowledge sources were to change substantially, the interface which rearranges the data into the expected format could be easily inserted at this control point. To receive the data from the lower level processors, GroupWords passes requests and control to the next control module in the hierarchy. That module is called GetWords.

The GetWords module is similar to the GroupWords module because it also does not directly invoke any knowledge source which analyzes text hypotheses. GetWords is responsible for formatting the OCR input prior to processing by the lower level knowledge sources. It uses a knowledge source embedded in a utility to limit the search depth of letter hypotheses. That knowledge source uses thresholds which are dependent upon the decision metrics of the particular OCR front-end being used. By isolating the front-end dependent processing from the rest of the expert system, it is easier to adapt this expert system to different optical character recognition front-ends. When enough OCR input is gathered, the data and control is passed to the last control module, WordExpert, for analysis by the lower level knowledge sources.

The WordExpert module is primarily responsible for appropriately invoking three knowledge source modules. The first of these modules is the spelling expert subsystem which includes many separate knowledge

sources concerning the sequential combination of characters. The other two knowledge source modules are only invoked if the spelling expert subsystem has favorable results. The later two knowledge sources do not eliminate any of the word hypotheses. Their effect is limited to modifying the weighting of those hypotheses which have already been approved by the spelling expert subsystem.

The control module hierarchy is summarized in Figure 1. The knowledge sources processing information at the whole sentences level have a unique control module. The knowledge sources processing information from letter sequences and whole words have a unique control module also. A separate control module is provided to accomplish data formatting between the lower level and upper level knowledge sources. Another control module is provided to accomplish the interface to the OCR front-end. This separation and modularity of control provides flexibility for step-wise system enhancement, for easy knowledge source replacement, and for interface to a variety of OCR front-ends.

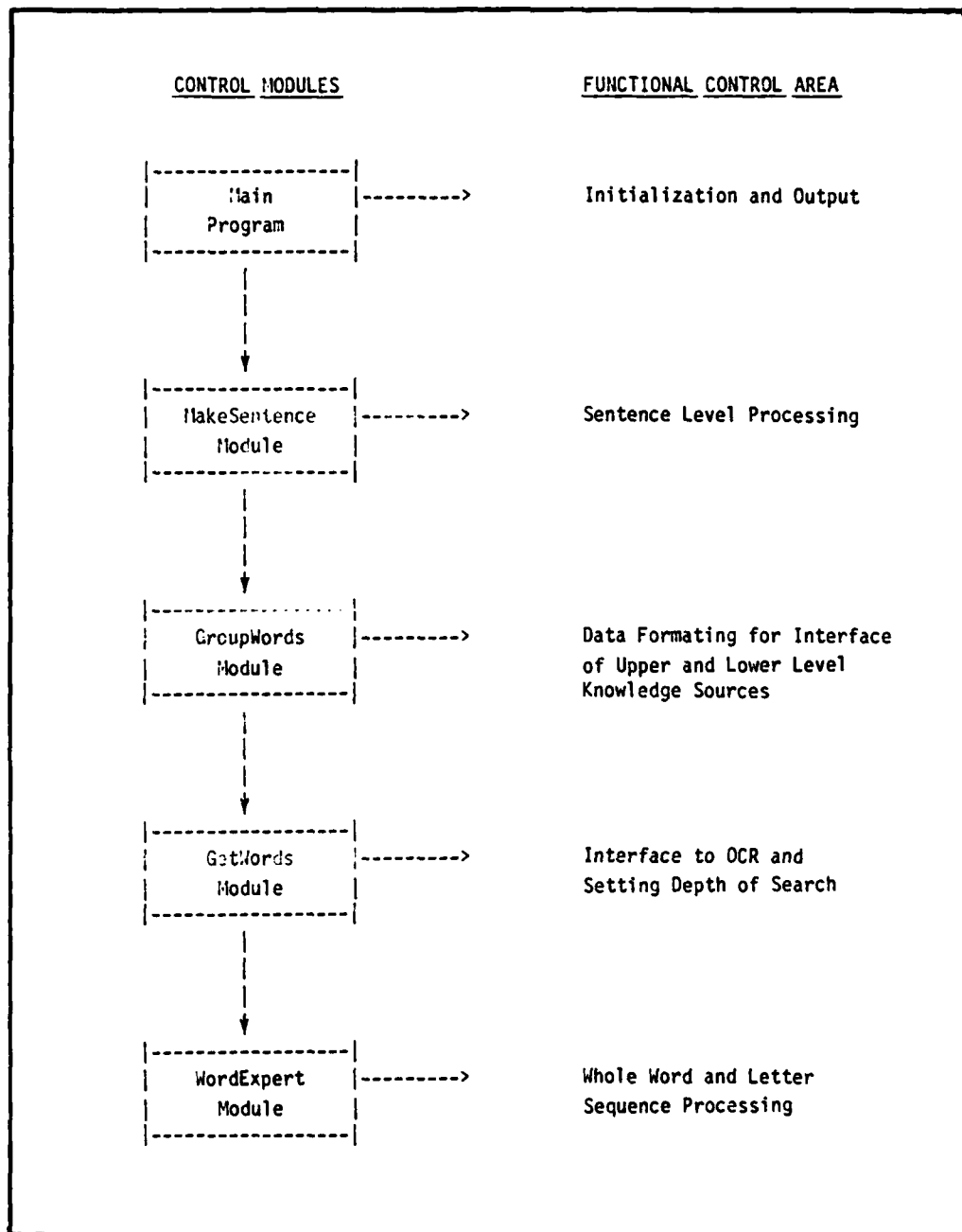


Figure 1. Control Module Hierarchy

Thresholding and Word Hypothesization.

The interface requirement imposed on the OCR front-end is that it be capable of supplying the postprocessor with a weighted list of possible characters for each character being recognized. If each letter position in a five letter word were assigned five possible characters, there would be up to 25 different word hypotheses. If the same was true for each word in a simple five word sentence, there would be up to 9,765,625 sentence hypotheses which would require evaluation by the postprocessor. It is very necessary to try to reduce the search space prior to any complex evaluation by the postprocessor. In order to eliminate the combinatorial explosion of processing that would occur if each letter position in a word was considered uncertain, a threshold is used to determine which letters can be assumed to have been accurately recognized by the OCR without postprocessing.

Many OCRs use thresholds to define the tolerance limit for accepting a match between the input data and the stored character prototypes. The thresholds may be established on an individual character basis or they can be uniform for the entire character set. Threshold assignments are dependent upon information particular to the measuring scheme employed by each OCR. Input data which does not match a character prototype within the tolerance specified by the thresholds are usually flagged by the OCR as unrecognizable characters. The alternative to using thresholds in an OCR is to use forced recognition. Forced recognition is a method that requires the OCR to select its best

guess for each character being recognized. There is no distinction, under a forced recognition algorithm, between characters which have accurate prototype matches and those which have inaccurate matches. The advantage of forced recognition is that it does not require as much operator assistance as a thresholding method. The trade off is that the forced recognition algorithm has a much higher rate of incorrect recognitions.

The higher automation efficiency of forced recognition OCRs could be achieved by thresholding OCRs without sacrificing accuracy if the characters flagged as unrecognizable by an OCR using thresholds were postprocessed using contextual information not included in the design of the OCR's matching prototypes. All characters recognized from data falling within the thresholds would have the same error rate using either the thresholding algorithm or the forced recognition algorithm. The improvement achieved by postprocessing the reject characters from the thresholding algorithm would come from overriding the parallel forced recognition decision when evidence derived from the natural constraints of the English language supported the decision.

This general approach has been shown successful in spoken word recognition systems operating in a limited domain. For instance, a 127 word vocabulary system designed to interpret requests for flight information and reservations showed an error rate improvement from the forced recognition error rate of 10.8% to an error rate of 0.4% by using thresholds and syntactic/semantic postprocessing (17: 1619-1623; 18). The address reading machines used for postal letter sorting also demonstrate the effectiveness of postprocessing within a limited domain

(19: 1032, 1041). The expert system designed in this thesis uses postprocessing techniques which attempt to avoid the restrictions of any particular subject domain. The expert system will use thresholds to place a feasible bound on the hypothesis search space to be postprocessed.

The thresholds should be selected by the analysis of historical data about the error rates for the particular OCR front-end that will be used. The thresholds should be set at a value which will obtain a recognition error rate, for the non-rejected characters, which is slightly better than the desired overall error rate. The characters rejected by the threshold will be postprocessed to improve the accuracy of the alternative forced recognition decision for those characters. Since the performance of the postprocessing is dependent upon the accuracy of the assumed known characters, the threshold must be set smaller than the threshold required to achieve the desired error rate among the non-rejected characters. The thresholds used for testing the expert system in this thesis were picked using a prototype distance matrix for a low frequency filtered 2D-FFT of a simple character set. The distance matrix and character set are found in Appendices B and C.

Spelling Expert Subsystem.

The spelling expert subsystem is the most essential component of this postprocessing system. The spelling expert is responsible for making a substantial reduction in the word hypothesis search space by

eliminating those hypotheses containing non-conforming letter sequences. The spelling expert subsystem is also embedded with control statements to activate supplementary knowledge sources which process information at the character sequence and word levels; these knowledge sources will be discussed in the next section of this chapter. The spelling expert is an original approach to representing the English language spelling constraints which are difficult to express in an explicit series of rules.

Spelling is a complex operation to master, even for human beings. Most people improve their spelling proficiency through their experience in using the language. The rules which may have been taught to us in school are few in number and are accompanied by exceptions. Even this very familiar spelling rule, taught to us in grammar school and included in texts on spelling (20:18), is accompanied by exceptions:

Put "i" before "e"
Except after "c"
Or when sounded like "a"
As in "neighbor" or "weigh".

Exceptions to this rule include:

ancient	height	forfeit	efficient
neither	weird	leisure	seize

Besides the problems of being an incomplete set and being applicable to only the most general cases, spelling rules usually reference the pronunciation of the word or word root when deciding between spelling options. The circumstances where word sound knowledge is required to use the spelling rules include knowledge of accented syllables, knowledge of the silent e, and 'sounds-like ...' knowledge

(20: 15 - 28). Using the traditional spelling rules as a basis, it would be very difficult to program all the required knowledge into an expert system and it is very unlikely that these rules would provide enough constraints on allowable letter sequences for the speller to be effective as the primary element of the postprocessor.

The spelling of a word is strongly linked to sounds used in speaking that word. However, it is not possible to construct a mapping between each of the individual letters and one or more specific sounds. This is a source of difficulty when using the phonic approach to teaching people to read (21: 156, 169, 183). Spelling is more closely patterned by a phonemic representation, where groupings of contrasting sounds can be represented by sequences of letters (21: 156, 169). Nevertheless, there is not a one for one mapping between separate phonemes and individual letters (21: 183). The mapping between phonemes and letter groups has a many to many correspondence. The indeterminism of these mappings makes it difficult to use the sound constraint techniques developed in speech recognition research as a basis for the spelling constraints of written words.

The rules used to develop the spelling expert for this postprocessor are based on several simple observations about spelling patterns:

1. Letter sequences can be separated into two major classes, vowels and consonants.
2. Each letter is a member of only one of these classes except for the letter, y. (Later, special provisions in the spelling algorithm, allow y to be classified as always a vowel.)

3. Regardless of how many phonemes are represented by a sequence of consonants or a sequence of vowels, words are composed of alternating groups of consonants and vowels.
4. A word can start with either a vowel sequence or a consonant sequence.
5. A word must always contain a vowel sequence (acronyms and abbreviations are not processed by the speller).
6. The sequences of vowels and the sequences of consonants that are allowed in a word can be sub-classed into those sequences which can occur in the beginning of a word, those which can occur in the middle of a word, and those which can occur at the end of a word.

A significant processing advantage for this representations is that it is much easier to separate sequences of consonants from sequences of vowels than to separate syllables based on pronunciation rules. One of the obstacles to this approach is the letter, y. This problem is solved by always classifying y as a vowel. The result of this classification is that the vowel sequences which included a consonant y, must include the vowels before and after that y in one continuous sequence. The lists of vowel sequences are not very long, so the inclusion of these additional multi-syllable sequences does not have a significant impact on search speed. However, the enumeration of the middle consonant sequences is not as easy to accomplish.

If completely enumerated, the list of middle consonant sequences would be quite long (approximately 2,400 entries). Instead, the list of middle sequences could be derived from a short list (only 36 entries)

of consonant sequences which may be used individually as a complete sequence or combined with the beginning consonant sequences (67 entries) to form a middle sequence. A separate module is used to find the appropriate split-up of the middle consonant sequence in a word so that two smaller consonant sequences can be matched, one each, against the middle and beginning consonant lists.

The processing flow of letter sequence checking is summarized in Figure 2. A word hypothesis is initially checked to verify the presence of a vowel sequence. If the word is completely composed of vowels, a special all-vowel sequence list is checked for a valid combination. Only those vowel sequences which form known words are included in that list. If the word hypothesis is not an all vowel sequence, the normal processing of alternating vowel and consonant sequences continues. The first letter sequence is matched against either the beginning vowel list or the beginning consonant list, whichever list is appropriate. If the first sequence was a consonant type, the next sequence is a vowel group and is matched against the middle vowel sequence list. If the first sequence was a vowel group then an initial attempt is made to match the following consonant group against the list of middle consonant sequences. If that attempt fails, the consonant sequence is split into two sequences, with the first sequence having a maximum length of three. The second sequence may initially be empty. Then, in successive trials until two matches are found, a letter is removed from the first sequence and pushed into the second sequence and new attempts are made to match these sequences against the beginning and middle consonant lists.

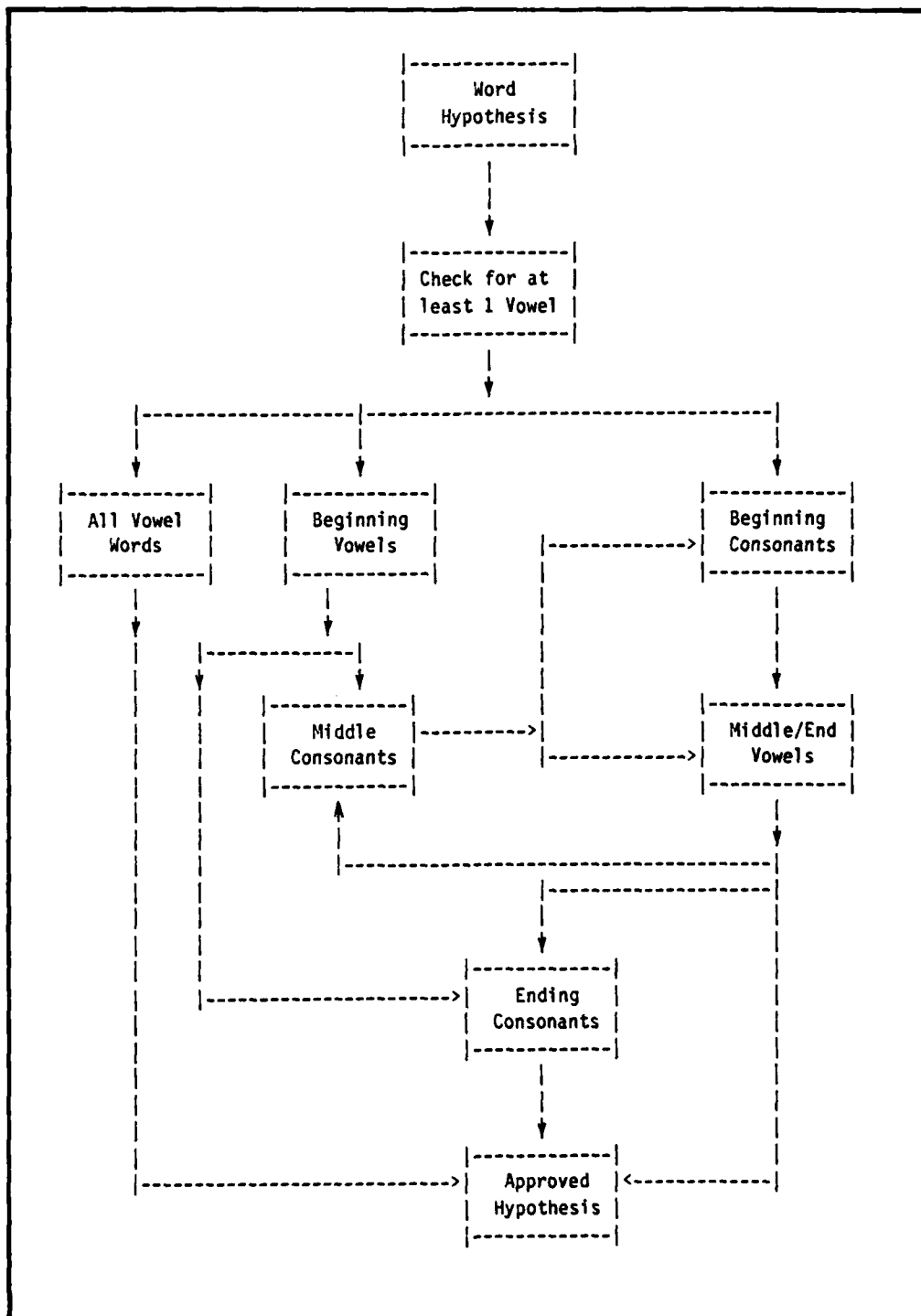


Figure 2. Letter Sequence Processing Flow

The matching process loop for middle vowel sequences and middle consonant sequences continues until the final vowel or consonant sequence in the word is found. If the final sequence in the word is a consonant group, it is matched against the ending consonant sequence list. However, ending vowel groups use the same sequence list as middle vowels, with the exception of the all-vowel words. Any sequence in a hypothesized word which fails the matching routines causes the word to be rejected as a hypothesis.

The decision to reject a word hypothesis can be overruled by the spelling expert subsystem. After the spelling routine is completed, a rejected hypothesis is checked against a list of known exceptions. For instance, the list of beginning vowel sequences does not include the double a; therefore, "aardvark" is included in a list of exceptions. Aardvark would be rejected by the spelling expert algorithm; but, the hypothesis rejection would be overruled by the knowledge source using the list of exceptions. The complete list of exceptions and unusual words can be found in the postprocessor program listings in Appendix A. Some of the words in the exceptions list may not seem as unusual as aardvark, but they contain a vowel or consonant sequence which is found in only a few known words. Their inclusion into the exceptions list results in additional constraints for spelling approval of a word hypothesis. The overall performance of the spelling expert subsystem is improved by using the exceptions list without compromising the bounds of the postprocessor vocabulary.

Supplementary Knowledge Sources.

There are several other knowledge sources operating within the framework of the spelling expert subsystem. These supplementary knowledge sources are included in the subsystem to either add more constraints to the letter sequence allowed in word hypotheses, or to demonstrate how specialized knowledge sources can be modularly included to handle unique processing difficulties that occur at the word level. Only two of the specialized knowledge sources are included to serve as examples of how extra processing rules and heuristics can enhance the postprocessor in a modular fashion. The two specialized knowledge sources are used to process words with apostrophes and words with hyphens.

The knowledge source which applies additional constraints to the selection of allowable letter sequences is very essential to increasing the effectiveness of the spelling expert. No constraints which can analyze letter sequences that include both consonants and vowels are implemented in the principle spelling expert. Also, the letter sequence constraints imposed by the principle spelling algorithm are very liberal because they allow a large amount of non-words to be accepted as hypotheses. That design feature was intentionally included to free the system vocabulary from any enumerable bounds. Those letter sequences, thought of as non-words under present day English, could become the newly coined words or variations on current words in the future versions

of our language. This supplementary knowledge source is a convenient place to add and remove spelling rules; especially as, English may be applicable to more and more rules as it continues its trend towards a more standardized lexicon (i.e. most new verbs have regular conjugation).

As a proposed spelling rule or spelling generalization approaches complete adherence by the language, that rule can be added to the supplementary spelling expert and the small number of exceptions to that rule can be included in a special lookup list. For instance, the spelling rule, "Q must be followed by U," is included in this knowledge source with its exceptions, such as qoph and qat, included in a list of unusual words. In contrast, if a spelling rule was proposed and included in the supplementary expert, but it was later discovered that there are wide spread exceptions to that rule, then the rule can be easily removed without disrupting the operation of the entire spell checker algorithm. A few of the other rules included in the supplementary expert are:

1. The sequence "pn" must be followed by the letters, "ue".
2. The sequence "ght" must be preceded by the letters, "au", "ou", or "i".

One other type of spelling rule is included in the supplementary expert. These rules are needed when the design of the principle spelling expert is not able to constrain an obviously incorrect sequence. It would be difficult to modify the principle algorithm directly. Therefore, this supplementary spelling expert is a convenient means to correct an unintentional, but inherent problem in the principle

spelling algorithm. The specific problem which appeared was the approval or word hypotheses which contained a triple l as a middle consonant sequence.

By using these supplementary rules, the entire spelling system becomes adaptive to our living and growing language. The rules implemented are not expected to be an exhaustive set for our current version of English. However, the inclusion of this implementation design provides the means for updating and improving the spelling expert subsystem as it is needed.

The two specialized knowledge sources, for hyphenated words and words with apostrophes, which are triggered within the principle spelling expert could have been placed under the control of the WordExpert control module. Although it was convenient to test the conditionals for these two supplementary knowledge sources during the data manipulation of the spelling system, the inclusion of additional supplementary modules within the speller is strongly discouraged. Embedding the control of additional modules into the spelling module could disrupt the cohesion of the spell checking process, especially when these supplementary modules are added, deleted, or modified.

The apostrophe expert functions by first locating the apostrophe within the word hypothesis. It then matches the contracted letters to a list of known possibilities. The root word is extracted and passed back to the spelling expert for further evaluation. The hyphenated word expert functions in a similar fashion. It first locates the hyphen within the word hypothesis in order to separate the word into two smaller words. After that, the hyphen expert recursively calls the

spelling expert for each of the two parts of the hyphenated word. After the second pass through the spelling expert, control is returned to the WordExpert module. If the first part of the hyphenated word is rejected by the speller, the second part of the hyphenated word is not processed.

Other supplementary knowledge sources could be modularly added to this system. The additional mini-experts could apply heuristics to make judgements on problems inherent to some texts. A simple example, which is used by some commercial reading machines, is deciding between a lower case l (el) and the numeral, 1 (one). These characters are identical in some fonts. The typical approach is to base the decision on the surrounding characters. If it is surrounded by other numerals, decide for the number, 1; likewise, if it is surrounded by letters, decide for the letter, l.

Another possibility for a supplementary knowledge source is an acronym expert. The acronym expert could possibly examine the first letters in the words preceding the acronym hypothesis. The expert could use one or more letters in each of the immediately preceding words to match one or more letters in the acronym hypothesis. This heuristic could be applied for the initial occurrence of an acronym. Discovered acronyms would have to be stored for matching against subsequent appearances of the acronym. The expert would be required to make choices between established acronyms and newly hypothesized acronyms. Implementation of this supplementary knowledge source could greatly enhance this postprocessor by removing one of the primary constraints placed on the input text. Unfortunately, non-trivial experts, such as the acronym expert, involve a significant design and programming effort,

and could not be included into the implementation studied under this thesis. Implementations of such experts can easily be appended to this postprocessor design and are left as a suggestion for future reading machine research projects.

Other Word Level Knowledge Sources.

There are two other knowledge sources which influence the weighting of competing word hypotheses. One knowledge source represents the vocabulary of the sentence parser used at the higher processing level. The other knowledge source represents a short term memory which is used to increase the belief weighting for words that are used frequently in text being processed. Both of these knowledge sources are implemented by simple dictionary lookup schemes.

The purpose of the knowledge source that represents the vocabulary of the sentence parser is to indirectly increase the belief weighting of sentences which can be processed by the parser. By performing the weighting at this level of processing, repetitive searches for the same word are avoided when the parsing operation takes place. Ideally, the part of speech, corresponding to the hypothesized word, would be stored in the data structure containing the word for later use by the parser. However, the parser operation is simulated in this postprocessor prototype; therefore, the parts of speech associated with each word are not included in the word hypothesis data structure or the dictionary file. The belief weightings of sentence hypotheses are increased in

proportion to the number of word in the sentence that are known to the parser. The belief measure of each word hypothesis found in the parsing dictionary is increased by constant factor. Thus, the belief measure for each sentence hypothesis, which is a combination of word belief measures, is influenced in proportion to the number of words known by the parser.

In a manner similar to that of the parsing dictionary knowledge source, the short term memory knowledge source influences the belief measures of word hypotheses. Again, a word hypothesis which is found in the lexicon of the short term memory has its belief measure increased. However, the aim of this knowledge source is to influence the belief measures of individual words, as opposed to the aim of influencing the corresponding sentence belief measures. Some of the words used within any body of text will have a natural tendency to recur. This is partially the reason why the frequency of individual word occurrences in the English language can be modeled by Zipf's Law (4; 12:52). The reason for calling this dictionary type of knowledge source a short term memory is because the size of the dictionary is a fixed length and it contains only the most recent words used in the text.

If the short term memory is reinitiated for each separate text being processed, the benefits of the short term memory would be appreciably small at the beginning of the text being processed. This effect can be offset by initializing the short term memory with a small amount of words. These words must be selected with minimum bias toward any particular subject domain in order to maintain consistent performance of the postprocessor for a wide variety of input texts.

There are several circumstances which cause some words to be repeated in a text more often than others. If the text concerns a specific subject domain, then words peculiar to that subject should appear quite often. The working vocabulary of the author will undoubtedly influence the words found in a specific text. These are good reasons for having the short term memory but they are not good foundations for picking the initialized memory. But there are other reasons for finding the repetition of words within a text. Some words are used repetitively in order to form transitions between thoughts expressed from one sentence to the next. Other words are repeated because they are necessary to format our thoughts within the syntax of the language (i.e. conjunctions, prepositions, and articles). Also, some words may be repeated because written thoughts are often expressed with respect to some common references about time, space, people, or object relationships (e.g. today, there, he, more). The words that are used repeatedly because of the manner in which English is naturally written should be the basis for initializing the short term memory.

If a very large sample of English texts were analyzed to determine which words occur most often, the words which appear at the top of that list should correspond to the words that appear in a most texts because of the nature of written English. Words which appear at the bottom of the frequency listing are likely to be there because of the particular subjects and authors that were sampled. In 1967, a study of word occurrence frequencies was performed on sample of English texts containing a total of 1,014,232 running words. The collection of texts was distributed among 500 samples of approximately 2,000 words. The

samples were chosen randomly from a wide variety of subject domains and prose styles with the exclusions of poetry and drama. Analysis of this collection of texts, known as the Standard Corpus of Present-Day Edited American English (also called the Brown Corpus after the university at which the study took place), enumerated exactly 50,406 different words. The ten most frequent words and their occurrence frequencies were (21):

the	69,971
of	36,411
and	28,852
to	26,149
a	23,237
in	21,341
that	10,595
is	10,099
was	9,816
he	9,543

The short term memory is initialized with the 200 most frequent words from the Brown Corpus. As sentences are approved by the postprocessor, the words which do not already appear in the short term memory are added. When the size limit of 400 words is reached, only the most recently used words are stored in the short term memory. The limit of 400 words was imposed to promote efficiency and prevent duplication of the parser dictionary effects.

Syntactic Expert.

The only sentence level analysis that is performed by the postprocessor is an analysis of the grammar used in the sentence. If a sentence hypothesis conforms to any of the arrangements of words allowed

by the rules of English syntax, the sentence hypothesis is permitted to compete for the final selection of the best sentence. Sentences which do not conform to proper syntax are eliminated from the solution search space. For sentences with words that are not in the parser's lexicon, the unknown words are treated as wild card parts of speech and only the known words determine whether the sentence structure conforms. The unknown words and their proposed parts of speech can be stored for future use by the parser. Prior to syntactic processing the group of sentence hypotheses are ranked in order of their relative measures of belief. The highest ranking sentence is processed by the parser first. If that sentence parses, no further processing by the parser is required. If no sentence in the group is approved by the syntax expert, the highest ranking sentence prior to the parsing attempt is forwarded as the best solution for the sentence. The words from this sentence are then used to update the short term memory and the parsing dictionary.

The implementation of the syntactic parser is simulated through an interactive input. The parsing vocabulary size was left undetermined. For test purposes, a word was considered to be in the parsing vocabulary if it was found in a standard dictionary of American English. No particular dictionary is cited because the testing of the postprocessor did not emphasize the vocabulary of the parser. Testing of the parser focused on its general contribution to the postprocessor accuracy.

IV. Testing and Results

Review of Goals.

The overall objectives of the English Sentence postprocessing system are to improve the accuracy and the throughput of reading machine technology. The postprocessor developed with this research is used to study how an expert system can apply diverse knowledge sources to reduce the uncertainty of optical character recognition and to remove any text input restrictions that are concerned with a fixed vocabulary or subject domain. Some of the specific questions that are examined are:

1. Are the knowledge sources which process text at a variety of levels, from characters on up to complete sentences (or higher), able to effectively combine evidence to produce a more accurate output?
2. Is the spelling expert able to operate effectively without imposing a limited vocabulary or presumed subject domain bias on the text input?
3. Are the interfaces to the sentence parser and to the OCR front-end designed to easily accept a wide variety of black box substitutions?

4. Is the Short Term Memory useful in improving the accuracy of the postprocessing system?

Testing Method.

Testing of the postprocessor was done by observing the processing results of some carefully selected input and some random input. The selective inputs were chosen with knowledge of the program design and expected program limitations to examine the general operation and the boundary conditions of the various algorithms and heuristics used in the postprocessor. The randomly selected input was used in an attempt to uncover some unexpected limitations of the program. Approximately, 2,000 individual words or psuedo-words were tested on the spelling algorithm and the entire system was tested with sentences totalling over 500 words. The amount of tests may seem small in comparison to the domain, but by directing the non-random portion of the tests at suspected weaknesses, the conditions which would reduce the effectiveness of the postprocessor were identified.

To perform the analysis of the postprocessing system, the program was embedded with screen output statements to observe the changes that take place for the size of the hypothesis search list and for the combination and modification of hypothesis weightings. In many cases, a reference to the knowledge source or the specific rule that was responsible for eliminating a hypothesis from the solution search list

was displayed to the terminal. A majority of the testing was focused on the performance of the spelling expert because of the novelty of its algorithm and because that component was needed to perform a significant reduction in the search space prior to the processing of other knowledge sources in the postprocessor.

Observations.

1. Are the knowledge sources which process text at a variety of levels, from characters on up to complete sentences (or higher), able to effectively combine evidence to produce a more accurate output?

Each of the knowledge sources used by the postprocessor effects the solution decision in one of two basic ways. A knowledge source can limit or reduce the number of competing solution hypotheses, or a knowledge source can modify the belief measure associated with each of the hypotheses. The following list categorizes the effects of the knowledge sources used in the postprocessor:

Thresholds - Restrict the number of characters competing for the same letter position within a word.

Apostrophe Expert - Reduces the number of competing word hypotheses if a rule violation occurs.

Hyphenation Expert - Reduces the number of competing word hypotheses if a rule violation is found by the spelling expert subsystem for any part of a hyphenated word.

Spelling Expert - Reduces the number of competing word hypotheses if a rule violation occurs for sequences of consonants or sequences of vowels within a word hypothesis.

Supplementary Spelling Rules - Reduces the number of competing word hypotheses if a rule violation occurs.

Exceptions Data Base - Overrides a word hypotheses reduction decision made by any of the above word level knowledge sources.

Vocabulary - Can modify the relative weightings of competing word hypotheses based on a dynamic parsing vocabulary.

Short Term Memory - Can modify the relative weightings of competing word hypotheses based on contextual recurrences.

Syntactic Parser - Reduces the number of competing sentence hypotheses if a violation of grammar rules occurs.

Whether the knowledge sources narrowed the solution search space or endorsed the plausibilities of particular hypotheses, the ability to improve the accuracy of the solution decision for each knowledge source could be associated with a general set of circumstances about the input data. Overall, the individual and the combined efforts of the knowledge sources in the expert system were useful in choosing the correct hypothesis for a large amount of test cases where the OCR front-end, employing a forced decision methodology, would have chosen incorrectly. Some examples of the performance improvement are shown in figures 3 and 4, on the next pages (sample text from mystery novel (23:108)). These particular test samples showed increases in the individual character recognition rates by 8.4% and 13.4%. The amount of improvement achieved through postprocessing varied with the input baseline accuracy. The circumstantial word and sentence ambiguities presented by text input also affected the recognition performance. Overall, it might be misleading to affix specific numbers to represent the character recognition improvement factors that were based only upon a collection

Forced Recognition: 90.4% accuracy.

There was no doubt in his mind. For some months he had found himself entertaining wild and melodramatic suspicions. He told me that he had been convinced that his wife was administering drugs to him. He had lived in India and the practice of wives driving their husbands insane by poisoning often comes up in the native courts. He had suffered fairly often from hallucinations with confusion in his mind about time and place.

Using Postprocessor: 98.8% accuracy.

There was no doubt in his mind. For some months he had found himself entertaining wild and melodramatic suspicions. He told me that he had been convinced that his wife was administering drugs to him. He had lived in India and the practice of wives driving their husbands insane by poisoning often comes up in the native courts. He had suffered fairly often from hallucinations with confusion in his mind about time and place.

Figure 3. Example of Postprocessing Improvement

Forced Recognition: 79.9% accuracy.

Thore mee no dcuht ln bis wind. Eor same montbs he had found hlmsclf cntertaining wiid anb welcdnematle sueyioions. He boid ma that he hed been aonvinced tbat his wife wcs administering dnugs to bim. Be had lived in India and tbe pnactioe of wives brivinyg lheir husbanhs insane bg poiaoniny often ccomes up in the retive oourts. Hc had soffered fairlg cften fnom ballocinabions with confusion ln his mind about time end qlace.

Using Postprocessor: 93.3% accuracy.

Thore mee no doubt in his wind. For same months he had found himself entertaining wild and welodnematle suepicions. He boid ma that he had been convinced that his wife was administering drugs to him. He had lived in India and the practice of wives briving their husbands insane by poisoniny often comes up in the retive courts. He had soffered fairly often from ballocinabions with confusion in his mind about time and place.

Figure 4. Example of Postprocessing Improvement

of tests which is very small relative to the possibilities that exist for generic text input data. Therefore, a qualitative analysis was done to describe the input data circumstances for which the expert system is unsuccessful in resolving the recognition uncertainty.

In general, the combination of evidence was necessary to resolve some uncertainties that could not be resolved by any individual knowledge source within the postprocessor. This is demonstrated through a summary of the processing steps for a simple test case. The actual sentence being postprocessed is, "The hat was brown."

The second word was input with two letter positions having uncertainty. The candidates for the h-position were b, h, and k. The candidates for the a-position were a, e, and c. These letter candidates resulted in nine word hypotheses: bat, hat, kat, bet, het, ket, bct, hct, and kct. The spelling algorithm eliminated the last three hypotheses immediately because they contained no vowels. The parsing vocabulary recognized bat, hat, and bet. Therefore, those words were given higher belief weightings over the other three nonsense words. The short term memory recognized the word, hat, from its use in a previous sentence. Therefore, the weighting of hat was increased so that it now ranked higher than bat, bet, or any of the other hypotheses. The sentence hypothesis containing hat was the first hypothesis examined by the parser. Since it conformed to acceptable grammar, the sentence was accepted as the solution. The combination of letter, word, and sentence level information was successful in choosing the correct hypothesis. However, there are circumstances which could produce the wrong solution if they are present in the data.

One data circumstance which heavily influences the eventual outcome is the quality of the input sensor data. If the characters being read were very noisy, the word level knowledge sources that increase the belief weightings for familiar words may not be able to compensate for a low weighting of the correct word hypothesis. In that situation, the unaided OCR would have made an incorrect decision too. An obvious extreme case of this problem occurs when the correct letter hypothesis is not within the top five choices selected by the thresholding module. For that situation, there is no avoiding an incorrect decision. The OCR must be sufficiently accurate that the correct choice is initially weighted within the hypothesis threshold.

There are circumstances where the unaided OCR could have made the correct choice, but it was led astray by the postprocessor. This situation is most notable for the short term memory knowledge source and will be discussed in more detail under question four. For cases where insufficient evidence is available to improve the decision, the hypothesis selection reverts back to the same information that is available to the unaided OCR. In the example used above, if the word, hat, was not recognized by the short term memory, the postprocessor and the unaided OCR may have chosen the word, bat. A human reader could not do any better unless high level semantic information, concerning the general subject of the text, was available.

Other situations where the postprocessor can inject errors into the recognition process occur when the expected input constraints are violated. The postprocessor has difficulty with accronyms, abbreviations, and some proper names. The erroneous hypothesis

rejections occur when these words do not conform to typical English phonemic structure. Another specialized knowledge source could be added to the postprocessor to handle the accronyms and abbreviations; a possible heuristic for designing this knowledge source is suggested in the next chapter. The difficulty with unusual proper names cannot be similarly resolved for a postprocessor which is designed not to be domain dependent. Another circumstance for extra errors occurs when the input sentences are not grammatically complete. If the actual sentence has incomplete grammar conformance, the parsing expert can select an incorrect sentence hypothesis that has an unknown word used as a wild card part of speech. Spelling mistakes in the input text are another violation of the prescribed input constraints that can cause errors to propagate from the postprocessor. Misspelled words can be accepted by the postprocessor when the misspelled word conforms to natural phonemic structure and it is not competing against a word hypothesis that is a common English word. Some misspelled words can even be corrected by the postprocessor if the incorrect letter was considered by the thresholds as an uncertain character. But the chances of this happening along with the other necessary conditions are very small. The type of misspelling which causes the most problems is the transposition of letters. Transpositions, especially within long consonant sequences, often produce sequences which are not allowed in natural English words. If a misspelled word is not rejected by the spelling expert, there is a good chance that the correct hypothesis will be slighted by the other knowledge sources in the postprocessor.

2. Is the spelling expert able to operate effectively without imposing a limited vocabulary or presumed subject domain bias on the text input?

The spelling expert did not exhibit any unintentional vocabulary restrictions during testing. Several types of test input data were used to validate the expected operation of the primary spelling algorithm. One test used words picked randomly from the dictionary and from assorted texts. Another test used words that were carefully selected to test all subcomponents of the spelling algorithm with emphasis on the splitting of middle consonant sequences. The last type of test data concentrated on the ability of the spelling algorithm to resolve the ambiguities of letters which typically have close prototype distances.

The input words which represent a random selection of valid English words were chosen from the dictionary and from a variety of texts such as newspaper articles, fiction, and technical journals. The dictionary words were chosen, several words in sequence, from a couple of random pages for each letter in the alphabet. The reason for picking several words in sequence from any page was to observe the differences in processing words that were similar in spelling. Approximately 1,000 dictionary words were tested with no test word rejected by the spelling expert. The test words, selected from assorted texts, were chosen in groups of three to five consecutive sentences (duplicate words were removed from the sample). Again, approximately 1,000 words were tested. However, one test word was rejected by the spelling algorithm. The proper name, McClellan, was rejected because of the beginning

consonant sequence, mccl. Further investigation showed that the spelling algorithm rejected a significant number of proper names. The rejections discovered were almost always for consonant sequences that never seemed to appear in English words that were not proper names. Many of the proper names which were rejected belonged to geographical locations in foreign countries such as Gdansk, Kwangchow, or Dneprodzerzhinsk. The names of people which were rejected were either names of people from foreign countries (e.g. Khrushchev) or names with strong ancestral ties to a non-English speaking country. The spelling algorithm could not be altered to accept these letter combinations without disrupting the model of typical English phonemic structure; most of the words which were rejected are very difficult to pronounce for the native English speaker. Also, the listing of all these proper names in the exceptions data base would make processing of the data base too time consuming. There are far too many name exceptions for people, places, and things to be listed. The only proper name that was included in the exceptions list the given name, John. The hn consonant sequence at the end of a word is unusual, but that name has very frequent occurrences in English text.

The next group of tests on the speller concentrated on the individual rules for each of the letter sequence knowledge sources. At least one word for each of the letter sequences in those knowledge sources was input to the spelling expert. This procedure validated that each of the rules was applicable to valid English words. Another phase of this group of testing was the input of nonsense words and of letter strings that purposely violated the natural English phonemic structure.

One sequence of middle consonants which violated the natural phonemic structure was identified. This structure, the tripple-l (el), was later corrected for by adding a rule to the supplementary spelling rule module in the spelling expert subsystem. All the tested nonsense words which did not conform to natural phonemic structure were rejected by the spelling algorithm.

The third area of testing on the spelling expert used data from the prototype distance matrix of a 2D-DFT of a simple character set. (See appendices B and C.) Characters which were more likely to be confused were suggested as letter hypotheses within a word to postprocessing system. The reactions of the spelling expert were then studied. For instance, the lower case letters, c and o, have very close prototypes for the test data. This information was used to set up competing word hypotheses. The input data for a word such as "color" might have an "o" substituted for the first letter to hypothesize the word, "ooler." If the spelling algorithm did not use an exceptions list for letter sequences which occur infrequently, the word, "ooler," would have been accepted by the algorithm. However, the few words which begin with a double-o are easy to enumerate from a dictionary (sequences in the middle of words are not as easy to list). Therefore, the word, "ooler," is rejected by the spelling expert. In most cases, the spelling expert did not need the spelling exceptions data base to reduce the hypothesis search space.

Although the prototype distance matrix will vary with the particular font being recognized and with the recognition method being used, the matrix indicates which letters are close enough in resemblance

to form words that may become sources of errors in the postprocessor. The closeness of the letters, c and o, could leave an unresolved ambiguity between the words, cat and oat. Another pair of letters which are reasonably close, h and b, can result in many ambiguous word pairs such as hit and bit, hat and bat, or hay and bay. Considering another close pair, g and y, a combination of two uncertain letter positions could cause a group of four ambiguous hypotheses: bay, bag, hay, and hag. This is not considered to be a shortcoming of the spelling algorithm; but, it does emphasize the need for additional knowledge sources in the contextual postprocessor to help resolve these ambiguities. Without the use of additional contextual knowledge, the performance of the spelling algorithm is predominantly determined by the signal quality of the input data for those circumstances where the close prototypes can form word hypotheses that have equal likelihood to any spelling expert.

An interesting situation occurs when an error exists for a character which is assumed by the thresholds as recognized with acceptable certainty. This error would be processed without notice by an unaided OCR using either a forced decision methodology or a thresholding scheme for flagging recognition rejections. However, there are numerous close letter prototypes that could draw attention if interchanged for the true letter. For example, any of the close prototypes which involve one consonant and one vowel could cause all words in a hypothesis group to be rejected by the spelling expert (i.e. if "c" was recognized instead of "o" when the vowel was the only vowel in the word). The postprocessor has some ability to recognize errors

that occur within the prescribed acceptance threshold for character recognition. On the other hand, those errors within the recognition threshold which are not recognized could propagate multiple letter errors within the word.

3. Are the interfaces to the sentence parser and to the OCR front-end designed to easily accept a wide variety of black box substitutions?

The interface requirements remained consistent with the original design specification. The OCR front-end must supply a weighted list of character hypothesis for each character position within a word. The only difficulty observed for changing the OCR front-end of the postprocessor concerned the setting of thresholds and the setting of the weighting factors used by the parsing vocabulary and short term memory knowledge sources. The thresholds can be computed from historical data about the OCR error rates. However, the weighting factors used by knowledge sources for changing the ranking of competing hypotheses are sensitive to accuracy and the thresholds of the OCR front-end. No formal relationship could be derived for the dependence of the weighting factors on the front-end accuracy. The general relationship which seemed appropriate was an inverse relationship; use smaller weighting factors for larger threshold values (more accurate front-ends). In the limit of front-end accuracy, the inverse relationship suggests that a very accurate OCR should not need the influence of contextual information to increase its accuracy or the very inaccurate OCR requires the influence of a lot of contextual information to improve accuracy.

4. Is the Short Term Memory useful in improving the accuracy of the postprocessing system?

In many test cases, the additional hypothesis weighting attributed to previous use of a word was a significant factor in the postprocessor choosing the correct hypothesis. In the example sentence used in question one, the final decision came down to just a couple highly ranked sentences. Given an equally weighted pair of sentences hypothesis such as "The hat was brown." and "The bat was brown.", the correct decision requires a high level semantic understanding of the text being processed. Without using any of the traditional approaches to story understanding (i.e. building an internal frame representation of the key concepts and relationships expressed in the text) the short term memory develops a dynamic, domain specific knowledge of the text. Its knowledge base is not as powerful as the domain specific semantic processors because it relies upon exact matching of key words as opposed to matching classes of words that could be inferred through a semantic network. For example, an advanced semantic processor could have chosen hat over bat by having an expectancy for an item of clothing and matching hat to the class of clothing items. The state of the art for semantic processing is not sufficient for operation in an unrestricted subject domain. The short term memory implemented in this postprocessor provides a few of the semantic processing benefits without the domain restrictions or processing overhead of traditional approaches to semantic processing.

In some situations, the semantic memory could incorrectly influence the postprocessor by increasing the weighting of a recurring word when that word was not related to the sentence currently being processed. These errors are kept to a small amount by limiting the dynamic portion of the short term memory to 200 words. Ideally, the 200 words would be distributed between words that are preferential in the author's vocabulary and words which are related to specific subject of the last few paragraphs of the text being processed. Texts which have frequent changes in subject matter and authors who introduce frequent changes in their writing style could produce the circumstances that allow the short term memory to incorrectly influence the postprocessor decisions. As the short term memory size is increased, the likelihood of errors is increased because the domain specialization developed by the short term memory is not appropriate for the text currently being processed.

V. Summary, Conclusions and Recommendations

Summary.

The hierarchical postprocessor is a suitable arrangement for improving the recognition error performance of reading machines operating in a forced recognition environment. The hierarchical architecture promotes a modular design; thus, permitting easy modification of the postprocessor knowledge sources and the ability to interface OCRs of a wide variety of identification methodologies. Each knowledge source used in the expert system was helpful in resolving text ambiguities through independent and cooperative processings of the solution hypotheses.

The most successful knowledge source of this contextual postprocessor is the unlimited vocabulary spelling algorithm. The algorithm employed rules about the sequences of consecutive vowels or consecutive consonants that conform to the natural phonemic structure of English words. Rejection of word hypothesis that do not conform to the rules of this algorithm resulted in a significant decrease in the solution search space that would be processed by other knowledge sources in the postprocessor. The spelling algorithm is considered to be vocabulary unlimited because it is liberal enough to accept some words that are not in the American English lexicon, as long as they have the appearance and sound of typical English syllables. Therefore, the

spelling algorithm is even capable of accepting words which may be coined in the future. The algorithm does accept nonsense word hypotheses; however, since those particular bad hypotheses are few in number they can be sorted out by the other knowledge sources in the expert system. The spelling algorithm rejects a majority of the incorrect hypothesis which permits more detailed processing to be performed on the remaining hypotheses.

Some of the other knowledge sources implemented in the expert system were concerned with assisting the spelling algorithm and with processing unusual text situations such as word contractions and hyphenated words. These knowledge sources were not intended to be inclusive of all the knowledge sources necessary to process generic text, but were intended to demonstrate the flexibility of this postprocessing system for modularly adding spelling rules, exceptions to rules, and other specialized knowledge sources that process very specific circumstances that occur in English text. Also, the format for a syntactic parser compatible with unlimited vocabulary operation within the postprocessor was defined.

Conclusions.

The accuracy of English sentence reading machines can be improved through a postprocessing expert system employing diverse knowledge sources. Arranging multiple knowledge sources in an hierarchical processing configuration is effective in reducing a significant amount

of the uncertainty inherent to the character identification process in OCRs. The use of many diverse knowledge sources is also effective in removing some of the constraints placed upon input texts.

The postprocessor developed by this research has demonstrated that a domain independent contextual postprocessor is producible and that it can effectively increase the accuracy and throughput of text reading machines. The key element to operation without domain dependent knowledge that would constrain the subjects of the input text is the unlimited vocabulary spelling algorithm. Also important to domain independent postprocessing is the use of wild card parts of speech for words which are not in the syntactic parser's vocabulary.

Recommendations.

A suitable continuation of this research effort could be the development of the syntactic parser needed for the sentence level processing. Established designs for English sentence parsers could be reconfigured to permit processing using wild card parts of speech. The parsing mechanism which handles the words not in the parsing vocabulary could also employ a truth maintenance scheme to increase the parser's lexicon after selection of sentence hypothesis. The truth maintenance process would have to be constrained to limit the processing time spent on backtracking and second guessing conclusions on older sentence hypotheses, especially in a long document.

Another area suitable for future research involves the spelling algorithm developed for this postprocessor. The spelling algorithm may

be useful in resolving letter segmentation problems. Its unique properties of operating on an unbounded vocabulary and quick decision processing may be very appropriate for reducing the identification ambiguities caused by letters merged together on the printed page or merged during the optical digitization process. The algorithm itself may not require any modifications. Instead, a group of competing input word hypothesis could have a variable length. For each word hypothesis, any letter position could be expanded to two or possibly three letter positions, depending upon the particular set of nonsegmented letters. The weighting process would also require adjustment to account for multiple letter sets competing against single letters in one group of word hypotheses.

This contextual postprocessing expert system shows great promise for the basis of a generic text reading machine. The general expansion of this system through the development of additional domain independent knowledge sources that are focused on removing the constraints presently placed upon input text data would be a productive step toward the development of an accurate, automated reading machine.

APPENDIX A

Postprocessor Program Listings and Data Base

This appendix contains the listing for the "reader.pas" postprocessing program and all the source files that are used to initialize the knowledge sources in the program. Those data base source files contain the beginning vowel sequences, beginning consonant sequences, middle consonant sequences, middle/end vowel sequences, completely vowel words, the 200 most frequent words, and the list of exceptions to the rule-based knowledge sources. Note that in the last listing, any "s" at the end of the word was removed to comply with the program requirements for finding plurals of words.

```
(*****  
(*
```

Post-Detection Processor
for
Expert System Reading Machine

By: Capt David V. Pacforkowski

December 1985

Description: This program accepts an input file of characters and their weightings from an OCR. Each line of the input file, Imagefile, contains alternating characters and real numbers (representing distance from the prototype) for each character position in the text being processed. A containing "9.9" flags the end of a word. A line containing a number greater than 10.0 flags the end of the text being processed. The only punctuation presently handled by this program are periods, question marks, exclamation marks, hyphens, and apostrophes. The syntactic parser is simulated with prompts to the operator. *)

```
(*****
```

```
program Reader (input, Imagefile, Vocabulary, Strange, Words200, BegSyl, MidSyl,  
                EndSyl, IsoVowels, Vowels, BegVowels, TempText, output);
```

```
type WordString = packed array [1..20] of char;  
    Syllable    = packed array [1..4] of char;  
    Image       = record  
        Character: char;  
        Measure  : real (* prototype distance / belief measure *)  
    end;  
    ImageList   = array [1..5] of Image;      (* up to 5 choices per letter *)  
    LetterList  = array [1..20] of ImageList; (* array of probable letters *)  
    WordGuess   = record  
        Word: WordString;      (* possible word *)  
        Belief: real           (* measure of belief for word *)  
    end;  
    WordList    = array [1..27] of WordGuess; (* group of possible words *)  
    WordGroup   = array [1..50] of WordList;  (* up to 50 words/sentence *)  
    Sentence    = record  
        SentGuess: array [1..50] of WordString;  
        Belief: real  
    end;  
    SentList    = array [1..20] of Sentence; (* array of probable sentences *)  
    SyllabPtr   = ^Syllable;                 (* pointer in linear list *)  
    SylList     = record  
        Syl : Syllable;  
        Next: SyllabPtr  
    end;  
    DictPtr     = ^DictWord;                  (* pointer in linear list *)  
    DictWord    = record  
        Word: WordString;  
        Next: DictPtr  
    end;
```

```

var Threshold : array[32..126] of real;
  Strange,      (* file of unusually spelled words *)
  Words200,     (* file of 200 most frequent words *)
  BegSyl,       (* file of beginning consonants *)
  MidSyl,       (* file of middle consonant letters *)
  EndSyl,       (* file of ending consonant letters *)
  Vowels,       (* file of middle/end vowel letters *)
  IsoVowels,    (* file of all vowel words *)
  BegVowels,    (* file of vowels beginning words *)
  TempText,     (* temp storage of read text *)
  Imagefile : text; (* contains input data from OCR *)
  StrangeWordList, (* points to unusual spellings *)
  ShortTermList, (* points to short term memory *)
  Beg200,       (* points to middle of s.t. memory *)
  IsoVowStart: DictPtr; (* points to all vowel word list *)
  BegSylStart,  (* points to start of a letter list *)
  MidSylStart,  (* points to start of a letter list *)
  EndSylStart,  (* points to start of a letter list *)
  BegVowStart,  (* points to start of a letter list *)
  VowelStart : SyllabPtr; (* points to start of a letter list *)
  NewSentence: Sentence; (* next sentence being read *)
  Done : boolean; (* indicates end of imagefile *)

```

(*****)

```

procedure Initialize;

```

```

var NewWord, Temp: DictPtr;
    NewSyl, Temp2: SyllabPtr;
    I: integer;

```

```

begin

```

```

  reset (Imagefile);
  Done := false;
  rewrite (TempText);

```

```

  (*----- Thresholds -----*)

```

```

  (* Set thresholds for recognizing character within the
     requirements for certainty. Set to just less than half the
     closest distance to next letter for tests in this thesis. *)

```

```

  for I := 32 to 126 do
    Threshold [I] := 0.2; (* default thresholds are .2 *)
  (* numerals: *)
  Threshold[48] := 0.08; Threshold[53] := 0.23;
  Threshold[49] := 0.00; Threshold[54] := 0.17;
  Threshold[50] := 0.29; Threshold[55] := 0.31;
  Threshold[51] := 0.20; Threshold[56] := 0.12;
  Threshold[52] := 0.19; Threshold[57] := 0.21;

```

```

(* upper case letters: *)
Threshold[65] := 0.25; Threshold[78] := 0.21;
Threshold[66] := 0.12; Threshold[79] := 0.12;
Threshold[67] := 0.17; Threshold[80] := 0.18;
Threshold[68] := 0.08; Threshold[81] := 0.34;
Threshold[69] := 0.19; Threshold[82] := 0.18;
Threshold[70] := 0.19; Threshold[83] := 0.22;
Threshold[71] := 0.18; Threshold[84] := 0.21;
Threshold[72] := 0.19; Threshold[85] := 0.18;
Threshold[73] := 0.06; Threshold[86] := 0.24;
Threshold[74] := 0.23; Threshold[87] := 0.39;
Threshold[75] := 0.28; Threshold[88] := 0.23;
Threshold[76] := 0.25; Threshold[89] := 0.24;
Threshold[77] := 0.36; Threshold[90] := 0.26;
(* lower case letters: *)
Threshold[97] := 0.23; Threshold[110] := 0.20;
Threshold[98] := 0.17; Threshold[111] := 0.11;
Threshold[99] := 0.11; Threshold[112] := 0.19;
Threshold[100] := 0.22; Threshold[113] := 0.17;
Threshold[101] := 0.21; Threshold[114] := 0.21;
Threshold[102] := 0.26; Threshold[115] := 0.27;
Threshold[103] := 0.16; Threshold[116] := 0.25;
Threshold[104] := 0.17; Threshold[117] := 0.20;
Threshold[105] := 0.31; Threshold[118] := 0.30;
Threshold[106] := 0.33; Threshold[119] := 0.39;
Threshold[107] := 0.23; Threshold[120] := 0.34;
Threshold[108] := 0.00; Threshold[121] := 0.16;
Threshold[109] := 0.36; Threshold[122] := 0.33;

(*----- Short Term Vocabulary -----*)
reset (Words200);
new (ShortTermList);
NewWord := ShortTermList;
while not eof (Words200) do
begin
  I := 1;
  NewWord^.Word := ' ';
  while not eoln (Words200) and (I < 21) do
  begin
    read (Words200, NewWord^.Word[I]);
    I := I + 1;
  end;
  readln (Words200);
  new (Temp);
  NewWord^.Next := Temp;
  NewWord := NewWord^.Next;
end;
Beg200 := NewWord;
Beg200^.Next := nil; (* mark end of linked list *)

```

```

(*----- Unusual Spellings -----*)
reset (Strange);
new (StrangeWordList);
NewWord := StrangeWordList;
while not eof (Strange) do
begin
  I := 1;
  NewWord^.Word := '          ';
  while not eof (Strange) and (I < 21) do
  begin
    read (Strange, NewWord^.Word[I]);
    I := I + 1
  end;
  readln (Strange);
  new (Temp);
  NewWord^.Next := Temp;
  NewWord := NewWord^.Next
end;
NewWord^.Next := nil; (* mark end of linked list *)

(*----- Beginning Syllables -----*)
reset (BegSyl);
new (BegSylStart);
NewSyl := BegSylStart;
while not eof (BegSyl) do
begin
  I := 1;
  NewSyl^.Syl := '          ';
  while not eof (BegSyl) and (I < 5) do
  begin
    read (BegSyl, NewSyl^.Syl[I]);
    I := I + 1
  end;
  readln (BegSyl);
  new (Temp2);
  NewSyl^.Next := Temp2;
  NewSyl := NewSyl^.Next
end;
NewSyl^.Next := nil; (* mark end of linked list *)

```

```

(*----- End Syllables -----*)
reset (EndSyl);
new (EndSylStart);
NewSyl := EndSylStart;
while not eof (EndSyl) do
begin
  I := 1;
  NewSyl^.Syl := ' ';
  while not eoln (EndSyl) and (I < 5) do
  begin
    read (EndSyl, NewSyl^.Syl[I]);
    I := I + 1;
  end;
  readln (EndSyl);
  new (Temp2);
  NewSyl^.Next := Temp2;
  NewSyl := NewSyl^.Next;
end;
NewSyl^.Next := nil; (* mark end of linked list *)

```

```

(*----- Middle Syllables -----*)
reset (MidSyl);
new (MidSylStart);
NewSyl := MidSylStart;
while not eof (MidSyl) do
begin
  I := 1;
  NewSyl^.Syl := ' ';
  while not eoln (MidSyl) and (I < 5) do
  begin
    read (MidSyl, NewSyl^.Syl[I]);
    I := I + 1;
  end;
  readln (MidSyl);
  new (Temp2);
  NewSyl^.Next := Temp2;
  NewSyl := NewSyl^.Next;
end;
NewSyl^.Next := nil; (* mark end of linked list *)

```

```

(*----- Vowel Groupings -----*)
reset (Vowels);
new (VowelStart);
NewSyl := VowelStart;
while not eof (Vowels) do
begin
  I := 1;
  NewSyl^.Syl := ' ';
  while not eoln (Vowels) and (I < 5) do
  begin
    read (Vowels, NewSyl^.Syl[I]);
    I := I + 1
  end;
  readln (Vowels);
  new (Temp2);
  NewSyl^.Next := Temp2;
  NewSyl := NewSyl^.Next
end;
NewSyl^.Next := nil; (* mark end of linked list *)

(*----- Vowel Words -----*)
reset (IsoVowels);
new (IsoVowStart);
NewWord := IsoVowStart;
while not eof (IsoVowels) do
begin
  I := 1;
  NewWord^.Word := ' ';
  while not eoln (IsoVowels) and (I < 5) do
  begin
    read (IsoVowels, NewWord^.Word[I]);
    I := I + 1
  end;
  readln (IsoVowels);
  new (Temp);
  NewWord^.Next := Temp;
  NewWord := NewWord^.Next
end;
NewWord^.Next := nil; (* mark end of linked list *)

```

```

(*----- Beginning Vowels -----*)
reset (BegVowels);
new (BegVowStart);
NewSyl := BegVowStart;
while not eof (BegVowels) do
begin
  I := 1;
  NewSyl^.Syl := ' ';
  while not eoln (BegVowels) and (I < 5) do
  begin
    read (BegVowels, NewSyl^.Syl[I]);
    I := I + 1
  end;
  readln (BegVowels);
  new (Temp2);
  NewSyl^.Next := Temp2;
  NewSyl := NewSyl^.Next
end;
NewSyl^.Next := nil (* mark end of linked list *)
(*-----*)

end; (* End of INITIALIZE module *)

(******)
procedure WordSearch (Start: DictPtr; TestWord: WordString; var Found: boolean);

(* Utility to use the parsing dictionary and the list of word exceptions to
the spelling algorithm. Linear search of linked list.*)

var Current: DictPtr;

begin

  Current := Start;
  Found := false;
  (*----- Search until match or end of linked list -----*)
  while (Current <> nil) and (not Found) do
    if Current^.Word = TestWord
    then Found := true
    else Current := Current^.Next
  (*-----*)

end; (* End of WORDSEARCH module *)

```

```

(*****)
procedure ExtraTest (Word: WordString; TestSyl: Syllable; var Valid: boolean);

  (* Rule-based knowledge source for exceptions and additions to the main
     spelling algorithm. *)

var I, J: integer;
    Match: boolean;

begin
  writeln('ExtraTest:', TestSyl);
  Valid := false;
  Match := false;
  I := 1;
  (*---- Find the Matching Syllable ----*)
  while not Match do
    begin
      (*--- Match the 1st letter of TestSyl ----*)
      if Word[I] = TestSyl[1]
      then Match := true
      else I := I + 1;
      J := 2;
      (*--- Match remaining letters of TestSyl ---*)
      while Match and (J < 5) and (TestSyl[J] <> ' ') do
        if TestSyl[J] = Word[I+1+J]
        then J := J + 1
        else
          begin
            I := I + 1;
            Match := false;
          end
        end;
      end;
      (*----- Special Rules -----*)
      if (TestSyl = 'q ') and (Word[I+1] = 'u')
      then Valid := true;

      if (TestSyl = 'pn ') and (Word[I+2] = 'u') and (Word[I+3] = 'e')
      then Valid := true;

      if (TestSyl = 'ght ') and ((Word[I-1] = 'u') and (Word[I-2] in ['a', 'o'])
                                or (Word[I-1] = 'i') and (Word[I-2] in ['e', 'b',
                                'f', 'l', 'm', 'n', 'r', 's', 't']))
      then Valid := true;

      if (TestSyl = 'll') and (Word[I+2] <> 'l')
      then Valid := true;

      if (TestSyl = 'zz') and (Word[I+2] <> 'z')
      then Valid := true
      (*-----*)
    end;
  end; (* End of EXTRA TEST module *)

```

```

(*****)
procedure Search (Start: SyllabPtr; Word: WordString; TestSyl: Syllable;
                 var Found: boolean);

    (* Linear search utility for spelling algorithm. Also used to trigger
       supplementary spelling exception rules. *)

var Current: SyllabPtr;

begin

    writeln ('Now searching for ', TestSyl);
    Current := Start;
    Found := false;

    (*----- Search until match or end of linked list -----*)
    while (Current <> nil) and (not Found) do
        if Current^.Syl = TestSyl
        then Found := true
        else Current := Current^.Next;

    (*----- Also check extra spelling rules -----*)
    if Found and ((TestSyl = 'q  ') or
                  (TestSyl = 'ght ') or
                  (TestSyl = 'll  ') or
                  (TestSyl = 'zz  ') or
                  (TestSyl = 'pn  '))
    then ExtraTest (Word, TestSyl, Found)
    (*-----*)

end; (* End of SEARCH module *)

(*****)
procedure SplitSearch (Word: WordString; First, Second: Syllable;
                      var Found: boolean);

    (* Utility to find appropriate split-up of middle consonant sequences; but
       does not correspond to pronounced syllable separations. *)

var Temp1, Temp2: Syllable;
    I, J, K, L: integer;

begin

    Found := false;
    Temp1 := First;
    Temp2 := Second;

    L := 3;
    for I := 4 downto 1 do
        if Temp2[I] = ' '
        then J := I;
    end for;

    (* counter for Temp1 letters *)
    (* find start of Temp2 blanks *)

```

```

for I := J to 3 do                      (* # of attempts = # of blanks *)
  if not Found
  then
    begin
      for K := (I-1) downto 1 do        (* shift letters *)
        Temp2[K+1] := Temp2[K];
      Temp2[1] := Temp1[L];             (* add new letter to Temp2 *)
      Temp1[L] := ' ';
      L := L - 1;                       (* increment counter *)
      if L <> 0                          (* check first consonant group *)
      then Search (MidSylStart, Word, Temp1, Found);
      if Found or (L = 0)                (* check second consonant group *)
      then Search (BegSylStart, Word, Temp2, Found)
    end
  end; (* End of SPLIT SEARCH module *)

(*****)
procedure Apostrophe (var Word: WordString; I: integer; var Length: integer;
  var Valid: boolean);

  (* Rule-based knowledge source for checking valid endings of apostrophe words;
  except for the word, o'clock, which is in exceptions listing. The word
  stripped of the apostrophe and ending is returned to the spelling
  algorithm for further processing. *)

var J: integer;

begin
  Valid := false;

  (*----- For the following cases: "...s" and "...s'" and "...d" and
  "...l]" and "...ve" and "...re"
  -----*)
  if ((Length = I) and (Word[I-1] = 's')) or
    ((Length = I + 1) and ((Word[I+1] = 's') or
      (Word[I+1] = 'd')) or
    ((Length = I + 2) and (((Word[I+1] = 'l') and (Word[I+2] = 'l')) or
      ((Word[I+1] = 'v') and (Word[I+2] = 'e')) or
      ((Word[I+1] = 'r') and (Word[I+2] = 'e'))))
  then
    begin
      Valid := true;
      for J := I to Length do
        Word[J] := ' ';
      Length := I - 1;
    end;
end;

```

```

(*----- Take care of "...n't" case -----*)
if ((Length = I + 1) and (Word[I-1] = 'n') and (Word[I+1] = 't'))
then
  begin
    Valid := true;
    for J := (I-1) to Length do
      Word[J] := ' ';
    Length := I - 1
  end;
(*-----*)

end; (* End of APOSTROPHE module *)

(*****)
procedure SpellCheck (NewWord: WordString; var WordValid: boolean);

  (* Main spelling algorithm - uses alternating sequences of vowels and
  consonants to accept word. *)

label 88, 99, 199;

type UpperCase = set of 'A' .. 'Z';
   LowerCase = set of 'a' .. 'z';

var Temp1, Temp2: Syllable;
    Length, I, J, K: integer;
    Found: boolean;
    Word, Word1, Word2: WordString;
    Capitals: UpperCase;
    Consonants, Vowels: LowerCase;

(*-----*)
begin

  Word1 := '          ';
  Word2 := '          ';

  (*----- Initialize Letter Sets -----*)
  Capitals := ['A' .. 'Z'];
  Vowels := ['a', 'e', 'i', 'o', 'u', 'y'];
  Consonants := ['a' .. 'z'] - Vowels;

```

```

(*----- Copy Word in Lower Case Letters -----*)
I := 1;
while (I < 21) and (NewWord[I] <> ' ') do
begin
  if NewWord[I] in Capitals
  then Word[I] := chr(ord(NewWord[I]) + 32)
  else Word[I] := NewWord[I];
  I := I + 1
end;
Length := I - 1; (* remember word length *)
writeln('Length = ',Length);
for I := (Length + 1) to 20 do (* fill word with blanks *)
  Word[I] := ' ';

(*----- Any single character word is Valid -----*)
if Length = 1
then WordValid := true;

(*----- Process Apostrophe & Hyphenated Words -----*)
for I := 1 to Length do (* check word for apostrophes *)
begin (* and for hyphenation *)
  (*-----*)
  if Word[I] = '''
  then
  begin
    Apostrophe (NewWord, I, Length, WordValid);
    writeln('Apostrophe check is ',WordValid);
    if not WordValid
    then goto 99
    else goto 88
  end;
  (*-----*)
  if Word[I] = '-'
  then
  begin (* Hyphen Word Processing *)
    for K := 1 to I-1 do
      Word1[K] := Word[K];
    for K := I+1 to Length do
      Word2[K] := Word[K];
    SpellCheck (Word1, WordValid);
    if WordValid
    then SpellCheck (Word2, WordValid);
    goto 199
  end
end;
88: (* exit for-loop after first apostrophe found *)

```

```

(*----- Find a Vowel in Word -----*)
WordValid := false;
for I := 1 to Length do
  if Word[I] in Vowels
    then WordValid := true;
if not WordValid
then
  begin
    writeln('No Vowel in word. ');
    goto 99
  end
else writeln ('Vowels were ok. ');

(*----- Check for "s" Pluralization -----*)
if Word[Length] = 's'
then
  begin
    Word[Length] := ' '; (* remove the "s" *)
    Length := Length - 1 (* shorten the word *)
  end;

(*----- Try to match an all Vowel Word -----*)
WordSearch (IsoVowStart, Word, WordValid);
if WordValid
then goto 99
else WordValid := true; (* reset for further processing *)

(*----- Process First Letter Group -----*)
Temp1 := ' ';
if (Word[1] in Vowels)
then (* collect first group of vowels & verify correctness *)
  begin
    I := 1;
    while Word[I] in Vowels do
      begin
        if I > 4 (* no more than 4 consecutive vowels *)
        then
          begin
            WordValid := false;
            writeln ('More than 4 vowels starting word ');
            goto 99
          end;
        Temp1[I] := Word[I];
        I := I + 1
      end;
    Search (BegVowStart, Word, Temp1, Found)
  end
end

```

```

else      (* collect first group of consonants & verify correctness *)
begin
  I := 1;
  while Word[I] in Consonants do
  begin
    if I > 4  (* no more than 4 consecutive starting consonants *)
    then
      begin
        WordValid := false;
        writeln ('More than 4 consonants starting word');
        goto 99
      end;
    Templ[I] := Word[I];
    I := I + 1
  end;
  Search (BegSylStart, Word, Templ, Found)
end;
J := I - 1;  (* remember letter position *)
if not Found (* verify first letter group was valid *)
then
  begin
    WordValid := false;
    writeln ('First letter group failed ',Templ, '|');
    goto 99
  end;

(*----- Process Remaining Syllables -----*)
while J < Length do
  begin
    writeln ('J=',J:2,'Length = ', Length:2);
    Templ := '   ';

    (*----- Get Next Letter Group -----*)
    if (Word[J+1] in Vowels)
    then
      begin
        I := J + 1;
        while Word[I] in Vowels do
        begin
          writeln ('vowel is:', Word[I]);
          if ((I - J) > 4)
          then
            begin
              WordValid := false;
              writeln('vowel group >4 failed: ',Word);
              goto 99
            end;
          Templ[I-J] := Word[I];
          I := I + 1
        end;
        Search (VowelStart, Word, Templ, Found);
        J := I - 1;  (* remember letter position *)
      end;
    end;
  end;

```

```

if not Found (* verify letter group was valid *)
then
begin
WordValid := false;
writeln ('Vowel group not found');
goto 99
end
end
else (*----- get next group, if consonants -----*)
begin
Temp2 := ' ';
I := J + 1; (* get next letter position *)
while (Word[I] in Consonants) and (I < 21) do
begin
if (I-J > 6) or ((I-J > 3) and (I-1 = Length))
then
begin
WordValid := false;
writeln ('Cons group greater than 6 letters. ');
goto 99
end;
if (I - J < 4) (* 1st attempt to split consonant syllables *)
then Temp1[I-J] := Word[I]
else Temp2[I-J-3] := Word[I];
I := I + 1
end;

if (I-1 = Length)
then
begin
Search (EndSylStart, Word, Temp1, Found);
J := I - 1
end
else
begin
Search (MidSylStart, Word, Temp1, Found);
if Found and (I-J > 4) and (I-1 < Length)
then (*--- attempt split search if enough consonants -----*)
begin
Search (BegSylStart, Word, Temp2, Found);
writeln ('Search Temp2:', Temp2);
end
else
if not Found and (I-1 < Length)
then
begin
writeln('SplitSearch Temp1:', Temp1, ' and Temp2:', Temp2);
SplitSearch (Word, Temp1, Temp2, Found)
end;
J := I - 1 (* remember last letter position *)
end;
end;

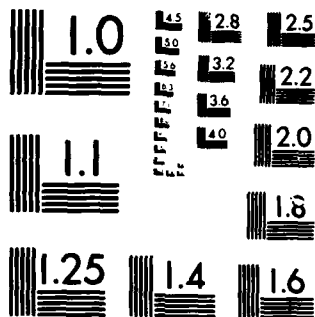
```

2/2

ML

FM MKD

©TNC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```

        if not Found          (*----- verify letter groups were valid *)
        then
            begin
                WordValid := false;
                goto 99
            end
        end
    end;
    (*----- Last Chance for Unusual Spellings -----*)
    if not WordValid
    then WordSearch (StrangeWordList, Word, WordValid)
    (*-----*)
99: (* Jump from hyphen routine or single-letter word *)

end; (* End of SPELLER module *)

(*****
procedure Normalize (Count: integer; var Choices: WordList);

    (* Utility used by WordExpert module to normalize the belief space for
    competing word hypotheses. *)

var I: integer;
    Sum: real;

begin
    Sum := 0.0;
    for I := 1 to Count do
        Sum := Sum + Choices[I].Belief;
    for I := 1 to Count do
        Choices[I].Belief := Choices[I].Belief / Sum
    end;

```

```

(*****>>>*****)
procedure WordExpert (Count: integer; var Choices: WordList);

  (* Lowest level control module. Used to interface word & letter level
  knowledge sources such as the spelling algorithm, the short term
  memory, and the parser's vocabulary. Additional rule-based knowledge
  sources can be added with calls from this module. *)

var I, J, NewCount, Position: integer;
    Found, WordValid: boolean;

begin

  (*----- Normalize Belief Measures -----*)
  Normalize (Count, Choices);

  (*-----*)

  (*---- Display current choices ----*)
  writeln('The choices for this word are:');
  for I := 1 to Count do
    writeln(Choices[I].Word, ' rated at ', Choices[I].Belief);
  *-----*)

  NewCount := Count;
  Position := 1;
  while Position <= NewCount do
    begin
      writeln('          Spell Checker Results');
      SpellCheck (Choices[Position].Word, WordValid);
      writeln (Choices[Position].Word:24, WordValid:8);
      readln;
      if not WordValid
      then (*----- shift other choices foward to eliminate word -----*)
        begin
          NewCount := NewCount - 1;
          for J := Position to NewCount do
            Choices[J] := Choices[J + 1];
          Choices[NewCount + 1].Word := '
        end
      else (*----- Check Parser's Dictionary If Word Valid -----*)
        begin
          WordSearch (StartDict, Choices[Position].Word, Found);
          writeln ('Dictionary Check = ', Found:8);
          if Found
          then Choices[Position].Belief := Choices[Position].Belief * 1.1;
        end
      end
    end
  end

```

```

        (*----- Check Short Term Memory If Word Valid -----*)
        WordSearch (ShortTermList, Choices[Position].Word, Found);
        writeln ('Short term memory check = ', Found);
        if Found
        then Choices[Position].Belief := Choices[Position].Belief * 1.1;
        Position := Position + 1
        end
    end;
    if NewCount = 0
    then writeln ('Threshold accepted characters are in Error.');
```

(*----- Normalize Belief Measures -----*)
 Normalize (NewCount, Choices);
 writeln('The finalized choices are:');
 for I := 1 to NewCount do
 writeln(Choices[I].Word, ' believed at ', Choices[I].Belief);
 end; (* End of WORD EXPERT control module *)

(*****
 procedure GetLetters (var NextLetters: ImageList; var EndWord: boolean);

(* Interface to input data from OCR. Assumes the depth of competing
 characters has been limited to five choices. *)

```

    var I: integer;
        Ch: char;
        Number: real;
        Stop: boolean;

    begin
        I := 1;
        Stop := false;
        while not eoln(Imagefile) and not EndWord and not eof(Imagefile) and
            not Stop do
            begin
                read (Imagefile, Ch, Number);
                writeln('The image input character is ', Ch, ' with vector distance = ',
                    Number);
                if (Number >= 9.8)
                then
                    begin
                        EndWord := true;
                        if (Number >= 10.0)
                        then Done := true
                    end
                end
            end
        end
    
```

```

else
begin
    if Number <= Threshold [ord(Ch)]
    then begin Number := 1.0; Stop := true end
    else Number := 0.1 / Number;
    NextLetters[I].Character := Ch;
    NextLetters[I].Measure := Number;
    I := I + 1
end;
if I < 6
then NextLetters[I].Character := ' '
end;
if not eof (Imagefile)
then readln (Imagefile)

end; (* End of GET LETTERS module *)
(*****
procedure ConnectLetters (LetterChoice: LetterList; var Combos: integer;
                        var WordChoices: WordList);

    (* Utility to perform letter combinametrics up to 3 uncertain characters
    per word. Called by the GetWords control module.*/)

var I, J, K, L, M, X, Y, Counter: integer;

(*----- Initialization Loop -----*)
begin
    for I := 1 to 27 do
    begin
        WordChoices[I].Word := '          ';
        WordChoices[I].Belief := 0.0
    end;
    I := 1;
    Combos := 1;
    (*----- Count # of Combinations -----*)
    while (I < 21) and (LetterChoice[I][1].Character <> ' ') do
    begin
        if LetterChoice[I][1].Measure < 1.0
        then
        begin
            J := 1;
            while (LetterChoice[I][J].Character <> ' ') do
                J := J + 1;
            Combos := Combos * (J - 1);
        end;
        I := I + 1
    end;
end;

```

```

I := 1;
(*----- Assemble Non-Variable Words -----*)
if Combos = 1
then
begin
WordChoices[2].Word[1] := ' ';
while (I < 21) and (LetterChoice[I][1].Character <> ' ') do
begin
WordChoices[1].Word[I] := LetterChoice[I][1].Character;
I := I + 1
end;
WordChoices[1].Belief := 1.0
end
(*----- Assemble Words from Letter Combinations -----*)
else
begin
Counter := 0;
while (I < 21) and (LetterChoice[I][1].Character <> ' ') do
begin
(*--- Copy Sure Letters to All Combos ---*)
if (LetterChoice[I][1].Measure = 1.0)
then
for J := 1 to Combos do (* fill array with sure letters *)
begin
WordChoices[J].Word[I] := LetterChoice[I][1].Character;
WordChoices[J].Belief := WordChoices[J].Belief + 1
end
(*--- Insert Unsure Letters into Appropriate Positions ---*)
else
begin
(*-- Collect Info for Letter Positioning --*)
Counter := Counter + 1;
J := 2; (* start at 2nd letter *)
while (LetterChoice[I][J].Character <> ' ') do
J := J + 1; (* J - 1 = # choices for this unsure letter *)
K := Combos div (J - 1); (* K = # choices - other unsure letters *)
if Counter = 1
then X := J - 1; (* remember for latter arrangements *)
if Counter = 2
then Y := J - 1; (* remember for latter arrangements *)
(*-----*)

if (Counter = 1) (* fill array for 1st unsure letter *)
then
for M := 0 to (J - 2) do
for L := (M * K + 1) to ((M + 1) * K) do
begin
WordChoices[L].Word[I] :=
LetterChoice[I][M + 1].Character;
WordChoices[L].Belief := WordChoices[L].Belief +
LetterChoice[I][M + 1].Measure
end;

```

```

        if (Counter = 2)          (* fill array for 2nd unsure letter *)
        then
            for M := 0 to (J - 2) do
                for L := (M * K + 1) to ((M + 1) * K) do
                    begin
                        WordChoices[L].Word[I] :=
                            LetterChoice[I][L mod (J-1)+1].Character;
                        WordChoices[L].Belief := WordChoices[L].Belief +
                            LetterChoice[I][L mod (J-1)+1].Measure
                    end;
                end;
            end;

        if Counter = 3          (* fill array for 3rd unsure letter *)
        then
            for M := 0 to (X * (J-1) - 1) do
                for L := 1 to Y do
                    begin
                        WordChoices[M*Y+L].Word[I] :=
                            LetterChoice[I][M mod (J-1)+1].Character;
                        WordChoices[M*Y+L].Belief := WordChoices[M*Y+L].Belief +
                            LetterChoice[I][M mod (J-1)+1].Measure
                    end
                end;
            end; (* end of else clause *)

            I := I + 1
        end
    end;
    WordChoices[Combos + 1].Word[1] := ' ';
    (*-----*)

end; (* End of CONNECT LETTERS module *)

(*****)
procedure GetWords (var WordChoices: WordList; var EndSentence: boolean);

    (* Control module to interface OCR and to set initial depth of search
       by using the OCR recognition thresholds. *)

var LetterChoices: LetterList;
    I, Combos: integer;
    NextLetter : ImageList;
    EndWord: boolean;

begin
    I := 1;
    EndWord := false;
    while not EndWord and (I < 21) do
        begin
            GetLetters (NextLetter, EndWord);
            LetterChoices[I] := NextLetter;
            I := I + 1
        end;
    end;

```

```

    if LetterChoices[I-2][1].Character in ['.', '?', '!']
    then EndSentence := true;
    ConnectLetters (LetterChoices, WordChoices, Combos);

    (*--- Process words with uncertain letters ---*)
    if (Combos > 1)
    then WordExpert (Combos, WordChoices)

end; (* End of GET WORDS control module *)

(*****)
procedure ConnectWords (NewWords: WordGroup; var NewSentences: SentList);

    (* Utility for performing sentence level combinametrics. Limited to
       three uncertain words per sentence. Called by GroupWords module.*)

var I, J, K, L, M, X, Y, Combos, Counter: integer;

begin
    Combos := 1;
    I := 1;
    while (I < 50) and (NewWords[I][1].Word[1] <> ' ') do
    begin
        if NewWords[I][1].Belief < 1.0
        then
            begin
                J := 1;
                while (NewWords[I][J].Word[1] <> ' ') do
                    J := J + 1;
                Combos := Combos * (J - 1)
            end;
        I := I + 1
    end;
    I := 1;
    if Combos = 1
    then
        while (I < 50) and (NewWords[I][1].Word[1] <> ' ') do
        begin
            NewSentences[1].SentGuess[I] := NewWords[I][1].Word;
            I := I + 1
        end
    end

```

```

else
begin
  Counter := 0;
  while (I < 50) and (NewWords[I][1].Word[1] <> ' ') do
  begin
    if (NewWords[I][1].Belief = 1.0)
    then
      for J := 1 to Combos do
      begin
        NewSentences[J].SentGuess[I] := NewWords[I][1].Word;
        NewSentences[J].Belief := NewSentences[J].Belief + 1
      end
    else
      begin
        Counter := Counter + 1;
        J := 2;
        while (NewWords[I][J].Word[1] <> ' ') do
          J := J + 1;
        K := Combos div (J - 1);
        if Counter = 1
        then X := J - 1;
        if Counter = 2
        then Y := J - 1;

        if Counter = 1
        then
          for M := 0 to (J - 2) do
            for L := (M * K + 1) to ((M + 1) * K) do
              begin
                NewSentences[L].SentGuess[I] := NewWords[I][M + 1].Word;
                NewSentences[L].Belief := NewSentences[L].Belief +
                  NewWords[I][M + 1].Belief
              end;
            end;

        if Counter = 2
        then
          for M := 0 to (J - 2) do
            for L := (M * K + 1) to ((M + 1) * K) do
              begin
                NewSentences[L].SentGuess[I] :=
                  NewWords[I][L mod (J-1)+1].Word;
                NewSentences[L].Belief := NewSentences[L].Belief +
                  NewWords[I][L mod (J-1)+1].Belief
              end;
            end;
          end;
        end;
      end;
    end;
  end;
end;

```

```

        if Counter = 3
        then
            for M := 0 to (X * (J-1) - 1) do
                for L := 1 to Y do
                    begin
                        NewSentences[M*Y+L].SentGuess[I] :=
                            NewWords[I][M mod (J-1)+1].Word;
                        NewSentences[M*Y+L].Belief := NewSentences[M*Y+L].Belief +
                            NewWords[I][M mod (J-1)+1].Belief
                    end
                end;
                I := I + 1
            end;
            for L := 1 to Combos do
                NewSentences[L].Belief := NewSentences[L].Belief /
                    (Combos * (I - 1) - (Counter * (Combos - 1)))
            end;
            NewSentences[Combos + 1].SentGuess[1][1] := ' '
        end; (* End of CONNECT LETTERS module *)

    (*****
    procedure GroupWords (var NewSentences: SentList);

    (* Mid-level control module. Used to interface word level processing to
    sentence level processing. If a upper/lower case knowledge source is
    added to this system, it may need the I=1 information from this module
    to mark the start of words. *)

    var NewWords: WordGroup;
        WordChoices: WordList;
        I: integer;
        EndSentence: boolean;

    begin
        I := 1;
        EndSentence := false;
        while not EndSentence do
            begin
                GetWords (WordChoices, EndSentence);
                NewWords[I] := WordChoices;
                I := I + 1;
            end;
            if I < 50
            then NewWords[I][1].Word[1] := ' ';
            ConnectWords (NewWords, NewSentences) (* makes all sentences combos *)
        end; (* End of GROUP WORDS control module. *)
    (*****

```

```

(*****)
procedure ParseSentence (var NewSentences: SentList;
                        var FinalChoice: Sentence);

(* This module simulates the syntactic parser of this postprocessing system.
   It relies on the operator to perform the grammar validation. If a word
   is not in the parser's vocabulary, it may take on any part of speech
   necessary to form a grammatically correct sentence. This module also
   selects the final sentence solution; it assumes sentence hypotheses are
   in order of preference. *)

var I, J, K: integer;
    Answer: char;
    Selected: boolean;

begin
    Selected := false;
    writeln (' ':20, 'Sentence Parser Simulation');
    writeln;
    I := 1;

    while (NewSentences[I].SentGuess[1][1] <> ' ') and not Selected do
    begin
        J := 1;
        writeln ('Does this sentence parse? (y/n)');
        writeln ('Allow unknown words to take on any necessary part of speech. ');
        repeat
            if (NewSentences[I].SentGuess[J] <> ' ')
            then write (NewSentences[I].SentGuess[J])
            else write (' ');
            J := J + 1;
        until (NewSentences[I].SentGuess[J - 1][1] in [' ', '.', '?', '!']);
        writeln;
        readln (Answer);
        if (Answer = 'y')
        then Selected := true;
        else I := I + 1;
    end;

    if Selected = true
    then FinalChoice := NewSentences[I]
    else FinalChoice := NewSentences[1]

end; (* End of PARSE SENTENCE module *)

```

```

(*****)
procedure StoreText (var FinalChoice: Sentence; var TempText: text);

  (* Utility used to record the final sentence choice in an output file. *)

var I, J: integer;

begin
  I := 1;
  repeat
    for J := 1 to 20 do
      if (FinalChoice.SentGuess[I][J] <> ' ')
        then write (TempText, FinalChoice.SentGuess[I][J])
        else write (TempText, ' ');
      I := I + 1;
    until (FinalChoice.SentGuess[I - 1][1] in [' ', '.', '?', '!']);
    writeln (TempText)

end; (* End of STORE TEXT module *)

(*****)
procedure RankSentences (var NewSentences: SentList);

  (* Utility used to normalize the sentence belief space and to place
  sentences in order of preference prior to parsing. *)

var I, J, K: integer;
    Sum: real;
    Temp: SentList;

begin
  I := 1;
  Sum := 0.0;
  while (NewSentences[I].SentGuess[1][1] <> ' ') do
    I := I + 1;
  if I = 2
  then NewSentences[1].Belief := 1.0
  else
    if I = 1
    then writeln('ERROR: No Sentence Parsed.')
    else
      begin
        for J := 1 to (I - 1) do
          Sum := Sum + NewSentences[J].Belief;
        for J := 1 to I do
          NewSentences[J].Belief := NewSentences[J].Belief / Sum;
      end
  end
end

```

```

    for J := 1 to (I - 1) do
    begin
        for K := (J + 1) to (I - 1) do
            Temp[J] := NewSentences[J];
            if (NewSentences[K].Belief > Temp[J].Belief)
            then
                begin
                    Temp[J] := NewSentences[K];
                    NewSentences[K] := NewSentences[J]
                end
            end;
        NewSentences := Temp
    end

end; (* End of RANK SENTENCE module *)

(*****)
procedure Remember (FinalChoice :Sentence);

    (* Utility used to update the short term memory and the parsing vocabulary
    after a sentence hypothesis is chosen. Since the parser dictionary is
    simulated, only a reminder to the operator is displayed for that portion
    of the update. The dynamic portion of the short term memory is limited to
    200 additional words. *)

var I, Count: integer;
    Current, Previous, Temp: DictPtr;
    Found: boolean;

begin

    writeln ('Please include all new words from chosen sentence into ',
            'parser dictionary. ');
    I:= 1;
    repeat
        Found := false;
        WordSearch (FinalChoice.SentGuess[I], Found);
        if not Found
        then
            begin
                new (Temp);
                Temp^.Word := FinalChoice.Sentguess[I];
                Temp^.Next := Begin200;
                Begin200 := Temp;
            end
        end;
        I:= I + 1;
    until I = Count + 1;
end;

```

```

(*-- remove duplicates -----*)
Found := false;
Previous := Begin200;
Current := Previous^.Next;
while (Current <> nil) and (not Found) do
  if Current^.Word = FinalChoice.SentGuess[I]
  then
    begin
      Previous^.Next := Current^.Next;
      dispose (Current)
    end;

(*-- Limit to 200 words -----*)
Count := 1
Current := Begin200;
while (Count < 201) and (Current^.Next <> nil) do
  begin
    Current := Current^.Next;
    Count := Count + 1
  end
  if Current^.Next <> nil
  then
    begin
      dispose (Current^.Next)
      Current^.Next := nil
    end
end
I := I + 1
until (FinalChoice.SentGuess[I - 1][1] in [' ', '.', '?', '!']);

end; (* End of the REMEMBER module *)

(*****)
procedure MakeSentence (var FinalChoice: Sentence; var TempText: text);

(* Higher level control module. Used to interface sentence level
rule-based knowledge sources such as the syntactic parser. Also used
to update the parsing vocabulary and the short term memory after each
sentence in the text completes processing. *)

var NewSentences: SentList;

begin
  GroupWords (NewSentences);
  ParseSentence (NewSentences);
  RankSentences (NewSentences);
  Remember (FinalChoice)

end; (* End of MAKE SENTENCE control module *)

```

```

(*****)
procedure Printout (var TempText: text);

  (* Utility used to display copy of output text file to terminal. *)
  var Ch: char;

begin
  writeln ('The completed script is:');
  reset (TempText);
  while not eof(TempText)do
    begin
      while not eoln(TempText) do
        begin
          read (TempText, Ch);
          write (Ch)
        end;
      readln (TempText);
      writeln
    end

end; (* End of PRINTOUT module *)

(*****)
begin (* MAIN PROGRAM *)

  Initialize;

  while not Done do
    begin
      MakeSentence (NewSentence);
      StoreText (NewSentence, TempText)
    end;
    Printout (TempText)

end. (* MAIN PROGRAM *)

(*****)

```

Beginning Vowel Sequences

a	eu	oi	yea
ae	i	ou	yi
au	io	u	yo
e	o	y	you
ea	oa	ye	yu

Middle/End Vowel Sequences

a	eo	oi	uey
ae	eou	oo	ui
aeo	eu	ou	uie
ai	ey	oy	uo
ao	eya	oye	uoi
aou	eye	oyi	uoy
au	i	oyou	y
ay	ia	u	ya
aya	ie	ua	ye
aye	io	uae	yea
ayo	iou	uai	yi
e	iu	uay	yie
ea	o	ue	yo
eau	oa	uea	you
ee	oe	uee	yu
ei			

All-Vowel Words

ay	ya
aye	yea
eau	you
eye	

Beginning Consonant Sequences

b	gl	ps	spr
bl	gn	q	sq
br	gr	r	st
c	h	rh	str
ch	j	s	sw
chl	k	sc	t
chr	kn	sch	th
cl	kr	scl	thr
cr	l	scr	tr
d	m	sh	tw
dr	mn	shr	v
dw	n	sk	w
f	p	sl	wh
fl	ph	sm	wr
fr	pl	sn	x
g	pn	sp	z
gh	pr	sph	

Middle Consonant Sequences

b	l	ng	st
c	lb	nk	t
ch	lf	p	tch
d	lk	r	v
f	ll	rch	w
ft	m	rd	wn
g	n	rth	x
ght	nch	s	z
h	nd	sh	zz
k			

Ending Consonant Sequences

b	ld	nsk	s
bb	lf	nt	sh
bt	lg	nth	sk
c	lk	nz	sm
ch	ll	p	sp
ck	lm	ph	st
cl	lp	pt	t
ct	lt	r	tch
d	m	rc	th
dg	mb	rch	tr
f	mm	rd	tt
ff	mn	rf	v
ft	mp	rg	w
g	mph	rk	wk
gh	n	rm	wl
ght	nc	rn	wn
gn	neh	rp	x
h	nd	rr	xt
hm	ng	rst	z
k	ngth	rt	zl
l	nk	rth	zz
lb	nn		

200 Most Frequent Words

the	we	man	get	however
of	him	me	here	home
and	been	even	between	small
to	has	most	both	found
a	when	make	life	thought
in	who	after	being	went
that	will	also	under	say
is	more	did	never	part
was	no	many	day	once
he	if	before	same	general
for	out	must	another	high
it	so	through	know	upon
with	said	back	while	school
as	what	years	last	every
his	up	where	might	don't
on	its	much	us	does
be	about	your	great	got
at	into	way	old	united
by	than	well	year	left
i	them	down	off	number
this	can	should	come	course
had	only	because	since	war
not	other	each	against	until
are	new	just	go	always
but	some	those	came	away
from	could	people	right	something
or	time	how	used	fact
have	these	too	take	though
an	two	little	three	water
they	may	state	states	less
which	then	good	himself	public
one	do	very	few	put
you	first	make	house	think
were	any	world	use	almost
her	my	still	during	hand
all	now	own	again	enough
she	such	see	without	far
there	like	men	place	took
would	our	work	american	head
their	over	long	around	yet

Exceptions to Rules

aardvark	eelgras	klaxon	oologic	qivuit
aardwolf	eelpout	kleenex	oologically	qoph
aeoliand	eelworm	kleptomania	oologist	queue
aeolic	een	kleptomaniak	oology	queuing
aeolipile	eer	klondike	oolong	schlemiel
aeon	eerie	kloof	oomiak	schlep
aeonian	eighth	klutz	oomph	schlieren
aorist	eocene	klystron	oophorectomie	schlimazel
aorta	eohippu	kohl	oophorectomy	schlock
aoudad	eolian	kohrabi	oophoriti	schmaltz
archaeology	eolith	kohrabie	oophyte	schmeer
archaeopteryx	eolithic	kvass	oosperm	schmo
archaeozoic	eon	kvetch	oospore	schmoose
ay	eonian	kvetche	oosporic	schmuck
ayah	eosin	kvetched	oosporou	schnapper
aye	eosinophil	kvetching	ootheca	schnapps
ayin	equiangular	kwacha	oothecae	schnecken
bdellium	eyas	llama	ootid	schnitzel
borscht	eyra	llano	ooze	schnook
buhl	eyrie	marshmellow	oozed	schnorrer
buhrston	eyrir	o'clock	oozier	schnozzle
chthonian	eyry	oedema	ooziest	schwa
chthonic	fahrenheit	oedipal	oozily	seeing
ctenidium	faience	oedipean	oozines	sixth
ctenoid	fellmonger	oedipu	oozing	sphragistic
ctenophore	fifth	oenology	oozy	thousandth
czar	fjeld	oenomel	oyer	tsar
czaravitch	fjord	oer	oyez	tsetse
czaravna	goeey	oersted	oyster	tsimme
czardas	grosz	oesophagus	phlebitis	tsunami
czarism	hundredth	oestrogen	phlebotomy	tsuri
czech	iactric	oestu	phlegm	tsutsugamushi
czechoslovakia	iacritical	oeuvre	phlegmatic	twelfth
czechoslovakian	iamb	oocyte	phloem	tzar
delft	iambic	oodles	phlogistic	tzimme
delftware	iambu	oogamete	phlogopite	tzuri
dhak	iasi	oogamie	phlogostron	uitlander
dharma	iatrogenic	oogamou	phlox	yacht
dhole	john	oogamy	qadi	yield
dhoti	khaki	oogenesi	qaf	zwieback
dhow	khan	oogenia	qaid	zwitterion
djinni	khat	oogenium	qasida	
djinny	klan	oolite	qat	
eau	klansman	oolith	qibla	
eel	klavern	oolitic	qintar	

APPENDIX B

Fourier Distance Matrix for Simple Character Set

This appendix contains the matrix of distances between the character feature sets for the characters shown in Appendix C. Each feature set was measured using the three lowest harmonics of a 2D-FFT on a 16 by 16 pixel image, using the program in Appendix D.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
A	0.00	0.55	0.68	0.57	0.68	0.69	0.61	0.57	1.03	0.85	0.79	0.85	0.97	0.64
B	0.55	0.00	0.61	0.48	0.49	0.58	0.60	0.43	1.02	0.69	0.74	0.76	0.84	0.63
C	0.68	0.61	0.00	0.46	0.65	0.77	0.37	0.65	1.10	0.66	0.75	0.67	1.01	0.75
D	0.57	0.48	0.46	0.00	0.56	0.66	0.46	0.47	1.14	0.56	0.76	0.62	0.98	0.62
E	0.68	0.49	0.65	0.56	0.00	0.40	0.68	0.50	1.05	0.72	0.76	0.63	0.93	0.70
F	0.69	0.58	0.77	0.66	0.40	0.00	0.32	0.49	1.17	0.84	0.77	0.89	0.93	0.70
G	0.61	0.60	0.37	0.46	0.68	0.82	0.00	0.63	1.10	0.75	0.76	0.63	1.05	0.65
H	0.57	0.43	0.65	0.47	0.50	0.49	0.63	0.00	1.21	0.70	0.55	0.67	0.87	0.43
I	1.03	1.02	1.10	1.14	1.05	1.17	1.10	1.21	0.00	1.07	1.19	1.28	1.16	1.22
J	0.85	0.69	0.66	0.56	0.72	0.84	0.75	0.70	1.07	0.00	0.87	0.76	1.09	0.81
K	0.79	0.74	0.75	0.76	0.76	0.77	0.76	0.55	1.19	0.87	0.00	0.75	0.92	0.61
L	0.85	0.76	0.67	0.62	0.63	0.89	0.63	0.67	1.28	0.76	0.75	0.00	1.03	0.73
M	0.97	0.84	1.01	0.98	0.93	0.93	1.05	0.87	1.16	1.09	0.92	1.03	0.00	0.95
N	0.64	0.63	0.75	0.62	0.70	0.70	0.65	0.43	1.22	0.81	0.61	0.73	0.95	0.00
O	0.53	0.52	0.36	0.26	0.61	0.70	0.41	0.54	1.11	0.60	0.82	0.67	0.99	0.67
P	0.59	0.45	0.65	0.54	0.57	0.49	0.74	0.47	1.16	0.82	0.72	0.83	0.81	0.73
Q	0.75	0.75	0.81	0.70	0.85	0.92	0.77	0.83	0.86	0.87	0.88	0.93	0.92	0.85
R	0.56	0.48	0.66	0.58	0.62	0.62	0.69	0.51	1.05	0.86	0.59	0.83	0.84	0.65
S	0.75	0.56	0.56	0.65	0.74	0.84	0.55	0.74	0.96	0.72	0.89	0.87	1.01	0.74
T	1.05	1.01	1.12	1.11	1.01	1.03	1.15	1.15	0.44	1.04	1.12	1.32	1.14	1.16
U	0.74	0.59	0.52	0.38	0.64	0.75	0.56	0.48	1.21	0.48	0.71	0.57	0.97	0.61
V	0.96	0.81	0.91	0.80	0.88	0.92	0.90	0.75	1.05	0.85	0.78	0.92	0.93	0.77
W	0.89	0.91	0.95	0.90	0.95	1.00	0.90	0.89	1.09	0.92	0.91	0.94	0.89	0.85
X	0.78	0.67	0.88	0.74	0.77	0.76	0.82	0.51	1.20	0.82	0.66	0.82	0.92	0.47
Y	1.08	0.86	1.05	0.99	0.94	0.99	1.07	0.89	0.92	0.96	0.93	1.07	0.92	0.88
Z	0.92	0.76	0.89	0.86	0.72	0.88	0.84	0.87	0.85	0.94	0.86	0.92	1.09	0.76
a	0.73	0.87	1.04	0.88	1.01	0.99	0.96	0.83	0.98	0.94	0.97	1.11	1.08	0.83
b	0.69	0.51	0.66	0.59	0.63	0.82	0.54	0.55	1.17	0.70	0.71	0.51	0.98	0.55
c	0.82	1.04	0.96	0.90	1.13	1.11	0.92	0.94	1.15	1.02	0.91	1.05	1.11	0.92
d	0.67	0.51	0.70	0.60	0.66	0.85	0.65	0.60	1.05	0.55	0.79	0.67	1.01	0.72
e	0.70	0.84	0.95	0.87	0.94	0.89	0.95	0.75	1.03	0.97	0.83	1.06	0.96	0.83
f	0.76	0.75	0.63	0.68	0.64	0.53	0.69	0.61	1.24	0.87	0.69	0.85	0.93	0.74
g	0.51	0.51	0.61	0.51	0.71	0.79	0.60	0.61	0.98	0.72	0.85	0.80	0.95	0.71
h	0.62	0.58	0.73	0.65	0.70	0.75	0.58	0.48	1.26	0.87	0.64	0.65	0.98	0.44
i	1.06	1.17	1.31	1.27	1.22	1.21	1.25	1.23	0.64	1.25	1.12	1.41	1.19	1.15
j	0.71	0.72	0.70	0.57	0.77	0.97	0.70	0.76	0.97	0.47	0.86	0.70	1.11	0.80
k	0.73	0.70	0.82	0.78	0.75	0.78	0.84	0.59	1.17	0.95	0.57	0.78	0.88	0.69
l	1.03	1.02	1.12	1.14	1.08	1.21	1.10	1.22	0.14	1.10	1.20	1.27	1.16	1.22
m	0.86	0.92	1.00	0.94	1.03	1.03	0.97	0.89	1.09	1.06	0.89	1.03	0.74	0.87
n	0.68	0.92	0.89	0.78	1.00	0.99	0.84	0.79	1.25	0.93	0.94	0.93	1.07	0.79
o	0.78	1.02	0.98	0.86	1.07	1.05	0.92	0.88	1.15	0.98	0.93	1.03	1.11	0.89
p	0.52	0.63	0.70	0.61	0.74	0.71	0.73	0.59	1.16	0.83	0.76	0.82	0.90	0.71
q	0.50	0.59	0.69	0.59	0.80	0.78	0.66	0.61	1.08	0.87	0.77	0.90	0.92	0.67
r	0.83	1.05	0.90	0.95	1.09	1.10	0.93	0.92	1.28	1.06	0.90	0.96	1.05	0.94
s	0.78	0.91	1.01	0.96	1.08	1.07	0.90	0.89	0.95	1.00	0.93	1.12	1.03	0.84
t	0.82	0.73	0.60	0.65	0.68	0.85	0.60	0.63	1.24	0.75	0.67	0.51	0.98	0.62
u	0.79	0.91	0.91	0.77	1.01	0.96	0.84	0.79	1.24	0.97	0.94	1.00	1.06	0.85
v	0.86	0.92	1.08	0.97	1.01	0.98	1.07	0.87	1.11	1.09	0.94	1.11	0.93	0.98
w	0.86	0.81	1.03	0.93	0.91	0.87	0.92	0.80	1.17	1.05	0.97	1.05	0.92	0.84
x	0.82	0.69	0.98	0.97	0.97	0.95	0.95	0.82	1.12	1.02	1.00	1.12	0.90	0.86
y	0.69	0.54	0.67	0.53	0.73	0.83	0.65	0.54	1.10	0.67	0.76	0.70	0.92	0.65
z	0.89	0.99	1.00	1.00	1.15	1.13	1.03	0.97	1.00	1.00	0.94	1.15	1.05	0.95
0	0.57	0.50	0.37	0.17	0.50	0.63	0.41	0.47	1.12	0.56	0.72	0.57	0.98	0.61
1	1.03	1.02	1.12	1.14	1.08	1.21	1.10	1.22	0.14	1.10	1.20	1.27	1.16	1.22
2	0.83	0.67	0.78	0.68	0.65	0.82	0.85	0.78	0.98	0.73	0.91	0.79	0.99	0.93
3	0.77	0.43	0.71	0.62	0.60	0.74	0.70	0.69	0.88	0.75	0.86	0.89	0.99	0.81
4	0.71	0.56	0.78	0.61	0.61	0.58	0.78	0.40	1.18	0.83	0.69	0.82	0.86	0.55
5	0.76	0.52	0.72	0.61	0.47	0.60	0.66	0.62	1.00	0.78	0.84	0.80	1.00	0.66
6	0.61	0.42	0.45	0.47	0.51	0.66	0.39	0.53	1.05	0.64	0.69	0.64	0.97	0.60
7	0.97	0.89	1.05	0.97	0.89	0.83	1.11	0.93	0.93	0.92	0.95	1.19	1.03	1.08
8	0.57	0.25	0.49	0.43	0.44	0.56	0.51	0.43	1.01	0.62	0.73	0.71	0.89	0.61
9	0.63	0.53	0.71	0.61	0.57	0.48	0.75	0.52	1.06	0.87	0.77	0.94	0.90	0.66

	O	P	Q	R	S	T	U	V	W	X	Y	Z	a	b
A	0.53	0.59	0.75	0.56	0.75	1.05	0.74	0.96	0.89	0.78	1.08	0.92	0.73	0.69
B	0.52	0.45	0.75	0.48	0.56	1.01	0.59	0.81	0.91	0.67	0.86	0.76	0.87	0.51
C	0.36	0.65	0.81	0.66	0.56	1.12	0.52	0.91	0.95	0.88	1.05	0.89	1.04	0.66
D	0.26	0.54	0.70	0.58	0.65	1.11	0.38	0.80	0.90	0.74	0.99	0.86	0.88	0.59
E	0.61	0.57	0.85	0.62	0.74	1.01	0.64	0.88	0.95	0.77	0.94	0.72	1.01	0.63
F	0.70	0.49	0.92	0.62	0.84	1.03	0.75	0.92	1.00	0.76	0.99	0.88	0.99	0.82
G	0.41	0.74	0.77	0.69	0.55	1.15	0.56	0.90	0.90	0.82	1.07	0.84	0.96	0.54
H	0.54	0.47	0.83	0.51	0.74	1.15	0.48	0.75	0.89	0.51	0.89	0.87	0.83	0.55
I	1.11	1.16	0.86	1.05	0.96	0.44	1.21	1.05	1.09	1.20	0.92	0.85	0.98	1.17
J	0.60	0.82	0.87	0.86	0.72	1.04	0.48	0.85	0.92	0.82	0.96	0.94	0.94	0.70
K	0.82	0.72	0.88	0.59	0.89	1.12	0.71	0.78	0.91	0.66	0.93	0.86	0.97	0.71
L	0.67	0.83	0.93	0.83	0.87	1.32	0.57	0.92	0.94	0.82	1.07	0.92	1.11	0.51
M	0.99	0.81	0.92	0.84	1.01	1.14	0.97	0.93	0.89	0.92	0.92	1.09	1.08	0.98
N	0.67	0.73	0.85	0.65	0.74	1.16	0.61	0.77	0.85	0.47	0.88	0.76	0.83	0.55
O	0.00	0.58	0.74	0.63	0.63	1.12	0.38	0.82	0.92	0.83	1.01	0.94	0.84	0.62
P	0.58	0.00	0.83	0.37	0.77	1.08	0.66	0.86	1.00	0.71	0.95	0.96	0.95	0.77
Q	0.74	0.83	0.00	0.67	0.83	0.87	0.83	0.84	0.80	0.89	0.90	0.85	0.83	0.86
R	0.63	0.37	0.67	0.00	0.72	0.99	0.71	0.72	0.86	0.67	0.85	0.78	0.90	0.70
S	0.63	0.77	0.83	0.72	0.00	0.99	0.73	0.90	0.99	0.80	0.90	0.68	1.04	0.68
T	1.12	1.08	0.87	0.99	0.99	0.00	1.17	0.98	1.10	1.13	0.85	0.82	0.99	1.21
U	0.38	0.66	0.83	0.71	0.73	1.17	0.00	0.70	0.93	0.73	0.91	0.95	0.87	0.55
V	0.82	0.86	0.84	0.72	0.90	0.98	0.70	0.00	0.91	0.82	0.50	0.85	0.85	0.86
W	0.92	1.00	0.80	0.86	0.99	1.10	0.93	0.91	0.00	0.92	1.05	1.02	0.92	0.84
X	0.83	0.71	0.89	0.67	0.80	1.13	0.73	0.82	0.92	0.00	0.79	0.84	0.83	0.70
Y	1.01	0.95	0.90	0.85	0.90	0.85	0.91	0.50	1.05	0.79	0.00	0.80	0.93	1.00
Z	0.94	0.96	0.85	0.78	0.68	0.82	0.95	0.85	1.02	0.84	0.80	0.00	1.10	0.80
a	0.84	0.95	0.83	0.90	1.04	0.99	0.87	0.85	0.92	0.83	0.93	1.10	0.00	0.95
b	0.62	0.77	0.86	0.70	0.68	1.21	0.55	0.86	0.84	0.70	1.00	0.80	0.95	0.00
c	0.83	1.02	0.91	0.97	1.13	1.15	0.86	0.86	0.98	1.02	1.08	1.22	0.60	1.05
d	0.63	0.74	0.81	0.69	0.73	1.11	0.59	0.89	0.83	0.71	0.97	0.88	0.86	0.46
e	0.79	0.36	0.86	0.82	1.05	1.02	0.82	0.78	0.95	0.91	0.92	1.12	0.48	0.97
f	0.68	0.61	0.93	0.70	0.82	1.14	0.74	0.96	1.01	0.83	1.08	0.98	1.10	0.89
g	0.45	0.62	0.71	0.57	0.65	1.01	0.57	0.71	0.92	0.84	0.90	0.89	0.81	0.63
h	0.68	0.72	0.88	0.61	0.77	1.24	0.66	0.89	0.83	0.59	1.04	0.86	0.90	0.36
i	1.27	1.22	0.92	1.06	1.16	0.65	1.33	1.05	1.08	1.09	0.95	0.97	0.83	1.28
j	0.64	0.87	0.81	0.85	0.73	1.03	0.63	0.94	0.95	0.95	1.03	0.86	0.93	0.68
k	0.80	0.65	0.82	0.47	0.97	1.15	0.77	0.76	0.81	0.65	0.90	0.93	0.86	0.72
l	1.12	1.18	0.84	1.06	0.97	0.50	1.21	1.04	1.08	1.20	0.91	0.87	0.96	1.16
m	0.92	0.92	0.84	0.89	1.00	1.11	0.93	0.93	0.89	0.94	0.98	1.10	0.80	0.99
n	0.69	0.93	0.98	0.97	1.05	1.27	0.71	0.96	1.04	0.97	1.14	1.21	0.65	0.91
o	0.79	1.00	0.89	0.96	1.14	1.15	0.82	0.83	0.97	1.01	1.07	1.22	0.53	1.02
p	0.56	0.44	0.83	0.52	0.84	1.13	0.64	0.80	0.98	0.79	0.99	1.03	0.84	0.75
q	0.56	0.57	0.77	0.51	0.75	1.05	0.66	0.75	1.01	0.81	0.93	0.92	0.83	0.73
r	0.85	0.99	1.09	1.02	1.12	1.31	0.86	1.06	1.14	1.06	1.18	1.28	0.89	1.05
s	0.89	1.00	0.88	0.91	0.89	0.98	0.92	0.80	0.95	0.89	0.89	1.04	0.56	0.97
t	0.66	0.87	0.89	0.81	0.79	1.25	0.55	0.89	0.90	0.82	1.04	0.88	1.08	0.52
u	0.69	0.89	0.90	0.86	1.04	1.24	0.75	0.77	0.94	0.96	1.07	1.21	0.63	0.94
v	0.93	0.86	0.91	0.76	1.12	1.12	0.95	0.67	1.03	1.00	0.89	1.14	0.79	1.04
w	0.91	0.94	0.96	0.96	1.05	1.17	0.93	1.05	0.82	0.94	1.12	1.12	0.77	0.86
x	0.91	0.83	1.04	0.84	0.80	1.14	0.95	0.98	1.04	0.79	0.92	1.05	0.81	0.87
y	0.52	0.68	0.77	0.62	0.72	1.12	0.45	0.58	0.91	0.75	0.80	0.90	0.83	0.55
z	0.95	0.97	0.90	0.90	1.03	1.02	0.97	0.92	0.97	0.83	0.90	0.82	0.60	1.08
0	0.23	0.55	0.73	0.58	0.63	1.11	0.36	0.80	0.91	0.74	0.99	0.83	0.91	0.57
1	1.12	1.18	0.84	1.06	0.97	0.50	1.21	1.04	1.08	1.20	0.91	0.87	0.96	1.16
2	0.77	0.63	0.83	0.65	0.77	0.98	0.82	0.95	1.00	0.77	0.92	0.54	1.07	0.84
3	0.70	0.69	0.78	0.67	0.53	0.88	0.79	0.91	1.01	0.78	0.86	0.62	1.00	0.72
4	0.66	0.60	0.88	0.61	0.78	1.10	0.59	0.66	1.02	0.71	0.79	0.83	0.90	0.72
5	0.67	0.76	0.84	0.72	0.59	0.95	0.71	0.84	1.02	0.82	0.89	0.82	1.04	0.63
6	0.48	0.62	0.79	0.57	0.53	1.05	0.54	0.85	0.88	0.71	0.98	0.71	0.96	0.42
7	1.04	0.78	0.93	0.80	0.98	0.76	1.07	1.03	1.08	0.84	0.95	0.61	1.03	1.14
8	0.44	0.48	0.76	0.51	0.45	1.01	0.54	0.82	0.91	0.63	0.86	0.71	0.89	0.54
9	0.64	0.53	0.84	0.55	0.67	0.95	0.72	0.78	1.06	0.72	0.82	0.73	0.93	0.84

	c	d	e	f	g	h	i	j	k	l	m	n	o	p
A	0.82	0.67	0.70	0.76	0.51	0.62	1.06	0.71	0.73	1.03	0.86	0.68	0.78	0.52
B	1.04	0.51	0.84	0.75	0.51	0.58	1.17	0.72	0.70	1.02	0.92	0.92	1.02	0.63
C	0.96	0.70	0.95	0.63	0.61	0.73	1.31	0.70	0.82	1.12	1.00	0.89	0.98	0.70
D	0.90	0.60	0.87	0.68	0.51	0.65	1.27	0.57	0.78	1.14	0.94	0.78	0.86	0.61
E	1.13	0.66	0.94	0.64	0.71	0.70	1.22	0.77	0.75	1.08	1.03	1.00	1.07	0.74
F	1.11	0.85	0.89	0.53	0.79	0.75	1.21	0.97	0.78	1.21	1.03	0.99	1.05	0.71
G	0.92	0.65	0.95	0.69	0.60	0.58	1.25	0.70	0.84	1.10	0.97	0.84	0.92	0.73
H	0.94	0.60	0.75	0.61	0.61	0.48	1.23	0.76	0.59	1.22	0.89	0.79	0.88	0.59
I	1.15	1.05	1.03	1.24	0.98	1.26	0.64	0.97	1.17	0.14	1.09	1.25	1.15	1.16
J	1.02	0.55	0.97	0.87	0.72	0.87	1.25	0.47	0.95	1.10	1.06	0.93	0.98	0.83
K	0.91	0.79	0.83	0.69	0.85	0.64	1.12	0.86	0.57	1.20	0.89	0.94	0.93	0.76
L	1.05	0.67	1.06	0.85	0.80	0.65	1.41	0.70	0.78	1.27	1.03	0.93	1.03	0.82
M	1.11	1.01	0.96	0.93	0.95	0.98	1.19	1.11	0.88	1.16	0.74	1.07	1.11	0.90
N	0.92	0.72	0.83	0.74	0.71	0.44	1.15	0.80	0.69	1.22	0.87	0.79	0.89	0.71
O	0.83	0.63	0.79	0.68	0.45	0.68	1.27	0.64	0.80	1.12	0.92	0.69	0.79	0.56
P	1.02	0.74	0.86	0.61	0.62	0.72	1.22	0.87	0.65	1.18	0.92	0.93	1.00	0.44
Q	0.91	0.81	0.86	0.93	0.71	0.88	0.92	0.81	0.82	0.84	0.84	0.98	0.89	0.83
R	0.97	0.69	0.82	0.70	0.57	0.61	1.06	0.85	0.47	1.06	0.89	0.97	0.96	0.52
S	1.13	0.73	1.05	0.82	0.65	0.77	1.16	0.73	0.97	0.97	1.00	1.05	1.14	0.84
T	1.15	1.11	1.02	1.14	1.01	1.24	0.65	1.03	1.15	0.50	1.11	1.27	1.15	1.13
U	0.86	0.59	0.82	0.74	0.57	0.66	1.33	0.63	0.77	1.21	0.93	0.71	0.82	0.64
V	0.86	0.89	0.78	0.96	0.71	0.89	1.05	0.94	0.76	1.04	0.93	0.96	0.83	0.80
W	0.98	0.83	0.95	1.01	0.92	0.83	1.08	0.95	0.81	1.08	0.89	1.04	0.97	0.98
X	1.02	0.71	0.91	0.83	0.84	0.59	1.09	0.85	0.65	1.20	0.94	0.97	1.01	0.79
Y	1.08	0.97	0.92	1.08	0.90	1.04	0.95	1.03	0.90	0.91	0.98	1.14	1.07	0.99
Z	1.22	0.88	1.12	0.98	0.89	0.86	0.97	0.86	0.93	0.87	1.10	1.21	1.22	1.03
a	0.60	0.86	0.48	1.10	0.81	0.90	0.83	0.93	0.86	0.96	0.80	0.65	0.53	0.84
b	1.05	0.46	0.97	0.89	0.63	0.36	1.28	0.68	0.72	1.16	0.99	0.91	1.02	0.75
c	0.00	1.06	0.47	1.01	0.88	1.00	1.01	0.98	0.92	1.14	0.80	0.48	0.24	0.87
d	1.06	0.00	0.94	0.95	0.65	0.60	1.21	0.55	0.73	1.05	1.01	0.93	1.01	0.74
e	0.47	0.94	0.00	0.94	0.78	0.92	0.93	0.97	0.77	1.02	0.75	0.55	0.44	0.79
f	1.01	0.95	0.94	0.00	0.84	0.82	1.26	0.95	0.87	1.28	0.95	0.94	1.00	0.74
g	0.88	0.65	0.78	0.84	0.00	0.69	1.14	0.65	0.77	0.98	0.90	0.80	0.82	0.47
h	1.00	0.60	0.92	0.82	0.69	0.00	1.25	0.85	0.58	1.25	0.95	0.87	0.97	0.70
i	1.01	1.21	0.93	1.26	1.14	1.25	0.00	1.17	1.10	0.66	1.01	1.21	1.03	1.18
j	0.98	0.55	0.97	0.95	0.65	0.85	1.17	0.00	0.95	0.97	1.00	0.88	0.95	0.80
k	0.92	0.73	0.77	0.87	0.77	0.58	1.10	0.95	0.00	1.17	0.88	0.96	0.92	0.69
l	1.14	1.05	1.02	1.28	0.98	1.25	0.66	0.97	1.17	0.00	1.07	1.24	1.14	1.17
m	0.80	1.01	0.75	0.95	0.90	0.95	1.01	1.00	0.88	1.07	0.00	0.79	0.81	0.86
n	0.48	0.93	0.55	0.94	0.80	0.87	1.21	0.88	0.96	1.24	0.79	0.00	0.42	0.77
o	0.24	1.01	0.44	1.00	0.82	0.97	1.03	0.95	0.92	1.14	0.81	0.42	0.00	0.81
p	0.87	0.74	0.79	0.74	0.47	0.70	1.18	0.80	0.69	1.17	0.86	0.77	0.81	0.00
q	0.85	0.75	0.78	0.79	0.36	0.68	1.13	0.79	0.75	1.08	0.35	0.76	0.80	0.39
r	0.56	1.06	0.66	0.94	0.96	1.00	1.24	0.99	0.98	1.28	0.81	0.44	0.62	0.86
s	0.57	0.98	0.56	1.03	0.81	0.93	0.81	0.96	0.95	0.93	0.73	0.67	0.60	0.89
t	0.39	0.72	0.94	0.74	0.81	0.63	1.34	0.77	0.78	1.24	0.95	0.86	0.99	0.90
u	0.47	0.99	0.56	0.92	0.76	0.87	1.16	1.03	0.86	1.24	0.85	0.55	0.42	0.78
v	0.78	1.01	0.62	1.03	0.77	0.98	1.00	1.10	0.77	1.11	0.86	0.86	0.71	0.74
w	0.99	0.87	0.85	0.95	0.96	0.81	1.15	1.09	0.95	1.17	0.80	0.89	0.94	0.96
x	1.03	0.85	0.84	1.05	0.88	0.83	1.09	1.09	0.87	1.11	0.87	0.93	1.06	0.90
y	0.88	0.62	0.78	0.86	0.33	0.63	1.21	0.66	0.70	1.08	0.88	0.80	0.82	0.55
z	0.68	0.96	0.67	1.12	0.96	1.02	0.84	0.99	0.79	0.99	0.79	0.85	0.76	0.93
0	0.91	0.59	0.87	0.61	0.53	0.63	1.27	0.59	0.75	1.14	0.94	0.78	0.87	0.59
1	1.14	1.05	1.02	1.28	0.98	1.25	0.66	0.97	1.17	0.00	1.07	1.24	1.14	1.17
2	1.22	0.66	1.11	0.91	0.78	0.90	1.16	0.69	0.81	0.99	1.07	1.18	1.19	0.76
3	1.18	0.69	1.02	0.85	0.72	0.83	1.11	0.72	0.92	0.88	1.06	1.13	1.17	0.90
4	0.99	0.80	0.78	0.70	0.63	0.69	1.22	0.87	0.74	1.18	0.87	0.84	0.92	0.67
5	1.15	0.79	1.00	0.76	0.68	0.72	1.18	0.80	0.91	1.02	1.06	1.03	1.11	0.88
6	1.04	0.49	0.97	0.69	0.57	0.48	1.22	0.67	0.73	1.07	0.98	0.93	1.02	0.66
7	1.20	0.97	1.06	0.93	1.00	1.11	0.92	0.97	0.95	0.97	1.08	1.27	1.19	0.94
8	1.04	0.52	0.87	0.68	0.56	0.60	1.18	0.68	0.72	1.02	0.93	0.91	1.02	0.66
9	1.04	0.88	0.33	0.62	0.67	0.78	1.12	0.91	0.79	1.08	0.93	0.94	1.00	0.72

	q	r	s	t	u	v	w	x	y	z	0	1	2	3
A	0.50	0.83	0.78	0.82	0.79	0.86	0.86	0.32	0.59	0.89	0.57	1.03	0.83	0.77
B	0.59	1.05	0.91	0.73	0.91	0.92	0.81	0.69	0.54	0.99	0.50	1.02	0.67	0.43
C	0.69	0.90	1.01	0.60	0.91	1.08	1.03	0.98	0.67	1.00	0.37	1.12	0.78	0.71
D	0.59	0.95	0.96	0.65	0.77	0.97	0.93	0.97	0.53	1.00	0.17	1.14	0.68	0.62
E	0.80	1.09	1.08	0.68	1.01	1.01	0.91	0.97	0.73	1.15	0.50	1.08	0.65	0.60
F	0.78	1.10	1.07	0.85	0.96	0.98	0.87	0.95	0.83	1.13	0.63	1.21	0.82	0.74
G	0.66	0.93	0.90	0.60	0.84	1.07	0.92	0.95	0.65	1.03	0.41	1.10	0.85	0.70
H	0.61	0.92	0.89	0.63	0.79	0.87	0.80	0.82	0.54	0.97	0.47	1.22	0.78	0.69
I	1.08	1.28	0.95	1.24	1.24	1.11	1.17	1.12	1.10	1.00	1.12	0.84	0.98	0.88
J	0.87	1.06	1.00	0.75	0.97	1.09	1.05	1.02	0.67	1.00	0.56	1.10	0.73	0.75
K	0.77	0.90	0.93	0.67	0.94	0.94	0.97	1.00	0.76	0.94	0.72	1.20	0.91	0.86
L	0.90	0.96	1.12	0.51	1.00	1.11	1.05	1.12	0.70	1.15	0.57	1.27	0.79	0.89
M	0.92	1.05	1.03	0.98	1.06	0.93	0.92	0.90	0.92	1.05	0.98	1.16	0.99	0.99
N	0.67	0.94	0.84	0.62	0.85	0.98	0.84	0.86	0.65	0.95	0.61	1.22	0.93	0.81
O	0.56	0.85	0.89	0.66	0.69	0.93	0.91	0.91	0.52	0.95	0.23	1.12	0.77	0.70
P	0.57	0.99	1.00	0.87	0.89	0.86	0.94	0.83	0.68	0.97	0.55	1.18	0.63	0.69
Q	0.77	1.09	0.88	0.89	0.90	0.91	0.96	1.04	0.77	0.90	0.73	0.84	0.83	0.78
R	0.51	1.02	0.91	0.81	0.86	0.76	0.96	0.84	0.62	0.90	0.58	1.06	0.65	0.67
S	0.75	1.12	0.89	0.79	1.04	1.12	1.05	0.80	0.72	1.03	0.63	0.97	0.77	0.53
T	1.05	1.31	0.98	1.25	1.24	1.12	1.17	1.14	1.12	1.02	1.11	0.50	0.98	0.88
U	0.66	0.86	0.92	0.55	0.75	0.95	0.93	0.95	0.45	0.97	0.36	1.21	0.82	0.79
V	0.75	1.06	0.80	0.89	0.77	0.67	1.05	0.98	0.58	0.92	0.80	1.04	0.95	0.91
W	1.01	1.14	0.95	0.90	0.94	1.03	0.82	1.04	0.91	0.97	0.91	1.08	1.00	1.01
X	0.81	1.06	0.89	0.82	0.96	1.00	0.94	0.79	0.75	0.83	0.74	1.20	0.77	0.78
Y	0.93	1.18	0.89	1.04	1.07	0.89	1.12	0.92	0.80	0.90	0.99	0.91	0.92	0.66
Z	0.92	1.28	1.04	0.88	1.21	1.14	1.12	1.05	0.90	0.82	0.83	0.87	0.54	0.62
a	0.83	0.89	0.56	1.08	0.63	0.79	0.77	0.81	0.83	0.60	0.91	0.96	1.07	1.00
b	0.73	1.05	0.97	0.52	0.94	1.04	0.86	0.87	0.55	1.08	0.57	1.16	0.84	0.72
c	0.85	0.56	0.57	0.99	0.47	0.78	0.99	1.03	0.88	0.68	0.91	1.14	1.22	1.18
d	0.75	1.06	0.98	0.72	0.99	1.01	0.87	0.85	0.62	0.96	0.59	1.05	0.66	0.69
e	0.78	0.66	0.56	0.94	0.56	0.62	0.85	0.84	0.78	0.67	0.87	1.02	1.11	1.02
f	0.79	0.94	1.03	0.74	0.92	1.03	0.95	1.05	0.86	1.12	0.61	1.28	0.91	0.85
g	0.36	0.96	0.81	0.81	0.76	0.77	0.96	0.88	0.33	0.96	0.53	0.98	0.78	0.72
h	0.68	1.00	0.93	0.63	0.87	0.98	0.81	0.83	0.63	1.02	0.63	1.25	0.90	0.83
i	1.13	1.24	0.31	1.34	1.16	1.00	1.15	1.09	1.21	0.84	1.27	0.66	1.16	1.11
j	0.79	0.99	0.96	0.77	1.03	1.10	1.09	1.09	0.66	0.99	0.59	0.97	0.69	0.72
k	0.75	0.98	0.95	0.78	0.86	0.77	0.95	0.87	0.70	0.79	0.75	1.17	0.81	0.92
l	1.08	1.28	0.93	1.24	1.24	1.11	1.17	1.11	1.08	0.99	1.14	0.00	0.99	0.88
m	0.85	0.81	0.73	0.95	0.85	0.86	0.80	0.87	0.88	0.79	0.94	1.07	1.07	1.06
n	0.76	0.44	0.67	0.86	0.55	0.86	0.89	0.93	0.80	0.85	0.78	1.24	1.18	1.13
o	0.80	0.62	0.60	0.99	0.42	0.71	0.94	1.06	0.82	0.76	0.87	1.14	1.19	1.17
p	0.39	0.86	0.89	0.90	0.78	0.74	0.96	0.90	0.55	0.93	0.59	1.17	0.76	0.90
q	0.00	0.89	0.81	0.87	0.76	0.72	0.97	0.88	0.49	0.95	0.60	1.08	0.86	0.81
r	0.89	0.00	0.78	0.89	0.78	0.96	1.06	1.00	0.96	0.84	0.92	1.28	1.22	1.23
s	0.81	0.78	0.00	1.02	0.65	0.76	0.93	0.75	0.82	0.67	0.96	0.93	1.14	1.02
t	0.87	0.89	1.02	0.00	0.97	1.08	0.95	1.02	0.72	1.09	0.59	1.24	0.96	0.37
u	0.76	0.78	0.65	0.97	0.00	0.64	0.89	0.93	0.72	0.85	0.78	1.24	1.15	1.10
v	0.72	0.96	0.76	1.08	0.64	0.00	1.04	0.93	0.73	0.92	0.96	1.11	1.06	1.11
w	0.97	1.06	0.93	0.95	0.89	1.04	0.00	0.86	0.95	1.06	0.94	1.17	1.15	0.98
x	0.88	1.00	0.75	1.02	0.93	0.93	0.86	0.00	0.88	0.79	0.97	1.11	1.02	0.89
y	0.49	0.96	0.82	0.72	0.72	0.73	0.95	0.88	0.00	0.96	0.54	1.08	0.30	0.76
z	0.95	0.84	0.67	1.09	0.85	0.92	1.06	0.79	0.96	0.00	1.00	0.99	1.01	1.08
0	0.60	0.92	0.96	0.59	0.78	0.96	0.94	0.97	0.54	1.00	0.00	1.14	0.67	0.63
1	1.08	1.28	0.93	1.24	1.24	1.11	1.17	1.11	1.08	0.99	1.14	0.00	0.99	0.88
2	0.86	1.22	1.14	0.96	1.15	1.06	1.15	1.02	0.80	1.01	0.67	0.99	0.00	0.63
3	0.81	1.23	1.02	0.87	1.10	1.11	0.93	0.89	0.76	1.08	0.63	0.88	0.63	0.00
4	0.60	0.97	0.39	0.71	0.34	0.81	0.90	0.33	0.56	1.06	0.61	1.18	0.90	0.75
5	0.76	1.18	1.02	0.70	1.03	1.06	0.96	0.97	0.71	1.20	0.59	1.02	0.84	0.51
6	0.61	1.03	0.97	0.62	0.93	1.03	0.87	0.87	0.58	1.04	0.40	1.07	0.69	0.57
7	0.99	1.30	1.08	1.22	1.18	1.04	1.17	1.07	1.04	0.97	0.97	0.97	0.64	0.81
8	0.64	1.02	0.92	0.67	0.91	0.98	0.86	0.72	0.58	0.95	0.39	1.02	0.60	0.41
9	0.64	1.05	0.92	0.85	0.90	0.90	0.96	0.85	0.71	1.01	0.59	1.08	0.80	0.59

	4	5	6	7	8	9
A	0.71	0.76	0.61	0.97	0.57	0.63
B	0.56	0.52	0.42	0.89	0.25	0.53
C	0.78	0.72	0.45	1.05	0.49	0.71
D	0.61	0.61	0.47	0.97	0.43	0.61
E	0.61	0.47	0.51	0.89	0.44	0.57
F	0.58	0.60	0.66	0.83	0.56	0.48
G	0.78	0.66	0.39	1.11	0.51	0.75
H	0.40	0.62	0.53	0.93	0.43	0.52
I	1.18	1.00	1.05	0.93	1.01	1.06
J	0.83	0.78	0.64	0.92	0.62	0.87
K	0.69	0.84	0.69	0.95	0.73	0.77
L	0.82	0.30	0.64	1.19	0.71	0.94
M	0.86	1.00	0.97	1.03	0.89	0.90
N	0.55	0.66	0.60	1.08	0.61	0.66
O	0.66	0.67	0.48	1.04	0.44	0.64
P	0.60	0.76	0.62	0.78	0.48	0.53
Q	0.88	0.84	0.79	0.93	0.76	0.84
R	0.61	0.72	0.57	0.80	0.51	0.55
S	0.78	0.59	0.53	0.98	0.45	0.67
T	1.10	0.95	1.05	0.76	1.01	0.95
U	0.59	0.71	0.54	1.07	0.54	0.72
V	0.66	0.84	0.85	1.03	0.82	0.78
W	1.02	1.02	0.88	1.08	0.91	1.06
X	0.71	0.82	0.71	0.84	0.63	0.72
Y	0.79	0.89	0.98	0.95	0.86	0.82
Z	0.83	0.82	0.71	0.61	0.71	0.73
a	0.90	1.04	0.96	1.03	0.89	0.93
b	0.72	0.63	0.42	1.14	0.54	0.84
c	0.99	1.15	1.04	1.20	1.04	1.04
d	0.80	0.79	0.49	0.97	0.52	0.88
e	0.78	1.00	0.97	1.06	0.87	0.83
f	0.70	0.76	0.69	0.93	0.68	0.62
g	0.63	0.68	0.57	1.00	0.56	0.67
h	0.69	0.72	0.48	1.11	0.60	0.78
i	1.22	1.18	1.22	0.92	1.18	1.12
j	0.87	0.80	0.67	0.97	0.68	0.91
k	0.74	0.91	0.73	0.95	0.72	0.79
l	1.18	1.02	1.07	0.97	1.02	1.08
m	0.87	1.06	0.98	1.08	0.93	0.93
n	0.84	1.03	0.93	1.27	0.91	0.94
o	0.92	1.11	1.02	1.19	1.02	1.00
p	0.67	0.88	0.66	0.94	0.66	0.72
q	0.60	0.76	0.61	0.99	0.64	0.64
r	0.97	1.18	1.03	1.30	1.02	1.05
s	0.89	1.02	0.97	1.08	0.92	0.92
t	0.71	0.70	0.62	1.22	0.67	0.85
u	0.84	1.03	0.93	1.18	0.91	0.90
v	0.81	1.06	1.03	1.04	0.98	0.90
w	0.90	0.96	0.87	1.17	0.86	0.96
x	0.88	0.97	0.87	1.07	0.72	0.85
y	0.56	0.71	0.58	1.04	0.58	0.71
z	1.06	1.20	1.04	0.97	0.95	1.01
0	0.61	0.59	0.40	0.97	0.39	0.59
1	1.18	1.02	1.07	0.97	1.02	1.08
2	0.90	0.84	0.69	0.64	0.60	0.80
3	0.75	0.51	0.57	0.81	0.41	0.59
4	0.00	0.57	0.70	1.01	0.57	0.44
5	0.57	0.00	0.53	1.01	0.49	0.51
6	0.70	0.53	0.00	0.97	0.35	0.66
7	1.01	1.01	0.97	0.00	0.87	0.84
8	0.57	0.49	0.35	0.87	0.00	0.48
9	0.44	0.51	0.66	0.84	0.48	0.00

APPENDIX C

Test Set of Simple Characters

This appendix contains the character set that was used to pick the groups of characters which would compete in a word hypothesis for the same character position. The character set was made of simple block style characters to represent a font which has little separation between characters in the n-dimensional space of the feature set representation used by an OCR.

A	B	C	D	E	F
G	H	I	J	K	L
M	N	O	P	Q	R
S	T	U	V	W	X
Y	Z	a	b	c	d
e	f	g	h	i	j
k	l	m	n	o	p
q	r	s	t	u	v

w x y z o l
2 3 4 5 6 7
8 9

APPENDIX D

2D-FFT Program Listing

This appendix list the pascal program used to perform a two-dimensional discrete fourier transform on the character set found in Appendix C. The program was also used to produce the distance matrix found in Appendix B. The program was written in Turbo Pascal to run on an IBM compatible microcomputer.

```

(*****)
(*)
      TWO-DIMENSIONAL DISCRETE FOURIER TRANSFORM

      This program performs two-dimensional discrete fourier transforms,
      2D-DFTs, on digitized arrays of 16 by 16 points. The input data is
      located in a file named, Pixels. This file is a list of real numbers,
      with one entry per line. The output file, Prototypes, is a matrix
      of distances between each of the 16 x 16 arrays processed. The 62
      arrays processed represent a digitized set of upper case letters,
      lower case letters, and the numerals, 0 through 9.
*)
(*****)
program LetterDFT (Pixels, Prototypes);

const  NumChar = 62;

type OneDArray = array [1..16] of real;
     TwoDArray = array [1..16, 1..16] of real;
     ProType   = array [1..49] of real;
     AllTypes  = array [1..NumChar] of ProType;
     DistArray = array [1..NumChar, 1..NumChar] of real;

var Reals, Imaginary: TwoDArray;
    Filtered: ProType;
    CharDFT: AllTypes;
    Matrix: DistArray;
    I, J, K, Line: integer;
    Infile, Outfile: text;

(*****)
(*) The procedures DFT and PFA perform a 2D-DFT by making two passes of a
    single dimension DFT with a transposition of rows and columns between
    passes. These procedures were adapted to run on Turbo Pascal from a
    prime factor program developed in Fortran by C. S. Burrus at Rice
    University (24: 127-135).
*)
(*****)
procedure DFT (var X, Y: TwoDArray);

var X1, Y1: OneDArray;
    J, K: integer;

(*****)
procedure PFA (var A, B: OneDArray);

const C161 = 0.70710678118654;
      C162 = 0.38268343236510;
      C163 = 1.30656296448763;
      C164 = 0.54119610014619;
      C165 = 0.92387953251128;

```

```

var R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14,
    R15, R16, T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11,
    S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14,
    S15, S16, U1, U2, U3, U4, U5, U6, U7, U8, U9, U10, U11: real;

```

```

begin

```

```

(*----- REAL EQUATIONS -----*)

```

```

R1 := A[1] + A[9];
R2 := A[1] - A[9];
R3 := A[2] + A[10];
R4 := A[2] - A[10];
R5 := A[3] + A[11];
R6 := A[3] - A[11];
R7 := A[4] + A[12];
R8 := A[4] - A[12];
R9 := A[5] + A[13];
R10 := A[5] - A[13];
R11 := A[6] + A[14];
R12 := A[6] - A[14];
R13 := A[7] + A[15];
R14 := A[7] - A[15];
R15 := A[8] + A[16];
R16 := A[8] - A[16];
T1 := R1 + R9;
T2 := R1 - R9;
T3 := R3 + R11;
T4 := R3 - R11;
T5 := R5 + R13;
T6 := R5 - R13;
T7 := R7 + R15;
T8 := R7 - R15;
T9 := (T4 + T8) * C161;
T10 := (T4 - T8) * C161;
R1 := T1 + T5;
R3 := T1 - T5;
R5 := T3 + T7;
R7 := T3 - T7;
R9 := T2 + T10;
R11 := T2 - T10;
R13 := T6 + T9;
R15 := T6 - T9;
T1 := R4 + R16;
T2 := R4 - R16;
T3 := (R6 + R14) * C161;
T4 := (R6 - R14) * C161;
T5 := R8 + R12;
T6 := R8 - R12;
T7 := C162 * (T2 - T6);
T8 := C163 * T2 - T7;

```

```

T9 := C164 * T6 - T7;
T10 := R2 + T4;
T11 := R2 - T4;
R2 := T10 + T8;
R4 := T10 - T8;
R6 := T11 + T9;
R8 := T11 - T9;
T7 := C165 * (T1 + T5);
T8 := T7 - T1 * C164;
T9 := T7 - T5 * C163;
T10 := R10 + T3;
T11 := R10 - T3;
R10 := T10 + T8;
R12 := T10 - T8;
R14 := T11 + T9;
R16 := T11 - T9;

```

(*----- IMAGINARY EQUATIONS -----*)

```

S1 := B[1] + B[9];
S2 := B[1] - B[9];
S3 := B[2] + B[10];
S4 := B[2] - B[10];
S5 := B[3] + B[11];
S6 := B[3] - B[11];
S7 := B[4] + B[12];
S8 := B[4] - B[12];
S9 := B[5] + B[13];
S10 := B[5] - B[13];
S11 := B[6] + B[14];
S12 := B[6] - B[14];
S13 := B[7] + B[15];
S14 := B[7] - B[15];
S15 := B[8] + B[16];
S16 := B[8] - B[16];
U1 := S1 + S9;
U2 := S1 - S9;
U3 := S3 + S11;
U4 := S3 - S11;
U5 := S5 + S13;
U6 := S5 - S13;
U7 := S7 + S15;
U8 := S7 - S15;
U9 := (U4 + U8) * C161;
U10 := (U4 - U8) * C161;
S1 := U1 + U5;
S3 := U1 - U5;
S5 := U3 + U7;
S7 := U3 - U7;
S9 := U2 + U10;

```

```

S11 := U2 - U10;
S13 := U6 + U9;
S15 := U6 - U9;
U1  := S4 + S16;
U2  := S4 - S16;
U3  := (S6 + S14) * C161;
U4  := (S6 - S14) * C161;
U5  := S8 + S12;
U6  := S8 - S12;
U7  := C162 * (U2 - U6);
U8  := C163 * U2 - U7;
U9  := C164 * U6 - U7;
U10 := S2 + U4;
U11 := S2 - U4;
S2  := U10 + U8;
S4  := U10 - U8;
S6  := U11 + U9;
S8  := U11 - U9;
U7  := C165 * (U1 + U5);
U8  := U7 - U1 * C164;
U9  := U7 - U5 * C163;
U10 := S10 + U3;
U11 := S10 - U3;
S10 := U10 + U8;
S12 := U10 - U8;
S14 := U11 + U9;
S16 := U11 - U9;

```

(*----- OUTPUT EQUATIONS -----*)

```

A[1] := R1 + R5;
A[9] := R1 - R5;
A[2] := R2 + S10;
A[16] := R2 - S10;
A[3] := R9 + S13;
A[15] := R9 - S13;
A[4] := R8 - S16;
A[14] := R8 + S16;
A[5] := R3 + S7;
A[13] := R3 - S7;
A[6] := R6 + S14;
A[12] := R6 - S14;
A[7] := R11 - S15;
A[11] := R11 + S15;
A[8] := R4 - S12;
A[10] := R4 + S12;
B[1] := S1 - S5;
B[9] := S1 + S5;
B[2] := S2 - R10;
B[16] := S2 + R10;

```

```

B[3] := S9 - R13;;
B[15] := S9 + R13;
B[4] := S8 + R16;
B[14] := S8 - R16;
B[5] := S3 - R7;
B[13] := S3 + R7;
B[6] := S6 - R14;
B[12] := S6 + R14;
B[7] := S11 + R15;
B[11] := S11 - R15;
B[8] := S4 + R12;
B[10] := S4 - R12;

end;

(*****
begin (* procedure DFT *)

(*----- Get Image Data -----*)

  for J := 1 to 16 do
    for K := 1 to 16 do
      begin
        readln (Infile, X[J,K]);
        Y[J,K] := 0.0
      end;
    end;
  end;

(*----- Do Row DFTs -----*)

  for J := 1 to 16 do
    begin
      for K := 1 to 16 do
        begin
          X1[K] := X[J,K];
          Y1[K] := Y[J,K]
        end;
        PFA (X1, Y1);
        for K := 1 to 16 do
          begin
            X[J,K] := X1[K];
            Y[J,K] := Y1[K]
          end
        end
      end;
    end;
  end;
end;

```

```

(*----- Do Column DFTs -----*)

for J := 1 to 16 do
  begin
    for K := 1 to 16 do
      begin
        X1[K] := X[K,J];
        Y1[K] := Y[K,J]
      end;
      PFA (X1, Y1);
      for K := 1 to 16 do
        begin
          X[K,J] := X1[K];
          Y[K,J] := Y1[K]
        end
      end;
    end;
  end;
end;

(*-----*)

(*****
The procedure Normalize processes the output which has been filtered
to include only the lowest three harmonics, and both the real and
imaginary data. The data is normalized so that all character prototypes
have uniform energy and can be mapped to the surface of a 49-dimensional
hypersphere of radius, 1.0. (See O'Hair thesis (3: 17))
*****)

procedure Normalize (var Vector: ProType);

var I: integer;
    Temp: real;
begin
  Temp := 0.0;
  for I := 1 to 49 do
    Temp := sqr (Vector[I]) + Temp;
  Temp := sqrt (Temp);
  for I := 1 to 49 do
    Vector[I] := Vector[I] / Temp
  end;
end;

```

```

(*****)
(* The procedure, Filter removes the real and imaginary components of the
   lowest three harmonics, and the DC component from the 16 by 16 output
   transform. *)
(*****)

```

```

procedure Filter (Reals, Imag: TwoDArray; var Vector: ProType);

```

```

begin

```

```

  Vector[1] := Reals[14][14];
  Vector[2] := Reals[14][15];
  Vector[3] := Reals[14][16];
  Vector[4] := Reals[14][1];
  Vector[5] := Reals[14][2];
  Vector[6] := Reals[14][3];
  Vector[7] := Reals[14][4];
  Vector[8] := Reals[15][14];
  Vector[9] := Reals[15][15];
  Vector[10] := Reals[15][16];
  Vector[11] := Reals[15][1];
  Vector[12] := Reals[15][2];
  Vector[13] := Reals[15][3];
  Vector[14] := Reals[15][4];
  Vector[15] := Reals[16][14];
  Vector[16] := Reals[16][15];
  Vector[17] := Reals[16][16];
  Vector[18] := Reals[16][1];
  Vector[19] := Reals[16][2];
  Vector[20] := Reals[16][3];
  Vector[21] := Reals[16][4];
  Vector[22] := Reals[1][14];
  Vector[23] := Reals[1][15];
  Vector[24] := Reals[1][16];
  Vector[25] := Reals[1][1]; (* DC term *)
  Vector[26] := Imag[1][2];
  Vector[27] := Imag[1][3];
  Vector[28] := Imag[1][4];
  Vector[29] := Imag[2][14];
  Vector[30] := Imag[2][15];
  Vector[31] := Imag[2][16];
  Vector[32] := Imag[2][1];
  Vector[33] := Imag[2][2];
  Vector[34] := Imag[2][3];
  Vector[35] := Imag[2][4];
  Vector[36] := Imag[3][14];
  Vector[37] := Imag[3][15];
  Vector[38] := Imag[3][16];
  Vector[39] := Imag[3][1];
  Vector[40] := Imag[3][2];
  Vector[41] := Imag[3][3];

```

```

Vector[42] := Imag[3][4];
Vector[43] := Imag[4][14];
Vector[44] := Imag[4][15];
Vector[45] := Imag[4][16];
Vector[46] := Imag[4][1];
Vector[47] := Imag[4][2];
Vector[48] := Imag[4][3];
Vector[49] := Imag[4][4];
end;

(*****)
(* The procedure, Neighbor computes the distances between all prototype
   vectors that were processed. These include 26 lower case letters, 26
   upper case letters, and 10 numerals. *)
(*****)
procedure Neighbor (CharDFT: AllTypes; var Matrix: DistArray);

var I, J, K: integer;
    Difference, Distance: real;

begin
    for K := 1 to NumChar do
        for I := 1 to NumChar do
            begin
                Distance := 0.0;
                for J := 1 to 49 do
                    Distance := Distance + sqr(CharDFT[I][J] - CharDFT[K][J]);
                Matrix[K][I] := sqrt (Distance);
                writeln (K:3, I:5)
            end
        end
    end;

(*****)

begin (*--- MAIN PROGRAM ---*)

    assign (Infile, 'Pixels');
    assign (Outfile, 'Prototypes');
    reset (Infile);
    rewrite (Outfile);

    for I := 1 to NumChar do
        begin
            writeln(I);
            DFT (Reals, Imaginary);
            Filter (Reals, Imaginary, Filtered);
            Normalize (Filtered);
            CharDFT[I] := Filtered
        end;
    end;

```

```

Neighbor (CharDFT, Matrix);
writeln (Outfile, 'Distance Matrix');
for K := 0 to 2 do
begin
  writeln (Outfile);
  for I := (K * 16 + 1) to ((K+1) * 16) do
  begin
    if (I < 27)
    then write (Outfile, char(I + 64):6);
    if (I > 26) and (I < 53)
    then write (Outfile, char(I + 70):6);
    if (I > 52)
    then write (Outfile, char(I - 5):6)
  end;
  writeln (Outfile);
  for I := 1 to NumChar do
  begin
    if (I < 27)
    then write (Outfile, char(I + 64));
    if (I > 26) and (I < 53)
    then write (Outfile, char(I + 70));
    if (I > 52)
    then write (Outfile, char(I - 5));
    for J := (K * 16 + 1) to ((K+1) * 16) do
      write (Outfile, Matrix[I][J]:6:2);
    writeln (Outfile)
  end
end;

writeln (Outfile);
for I := 49 to NumChar do
begin
  if (I > 26) and (I < 53)
  then write (Outfile, char(I + 70):6);
  if (I > 52)
  then write (Outfile, char(I - 5):6)
end;
writeln (Outfile);
for I := 1 to NumChar do
begin
  if (I < 27)
  then write (Outfile, char(I + 64));
  if (I > 26) and (I < 53)
  then write (Outfile, char(I + 70));
  if (I > 52)
  then write (Outfile, char(I - 5));
  for J := 49 to NumChar do
    write (Outfile, Matrix[I][J]:6:2);
  writeln (Outfile)
end;

writeln (Outfile);

```

```

(*-----*)
for I := 1 to NumChar do
begin
  if (I < 27)
  then writeln (Outfile,'Vectors for the letter:',char(I + 64));
  if (I > 26) and (I < 53)
  then writeln (Outfile,'Vectors for the letter:',char(I + 70));
  if (I > 52)
  then writeln (Outfile,'Vectors for the number:',char(I - 4));
  for J := 1 to 7 do
  begin
    Line := (J - 1) * 7;
    for K := 1 to 7 do
      write (Outfile, CharDFT[I][Line + K]:7:2);
    writeln (Outfile)
  end;
  writeln (Outfile)
end;

close (Outfile)

end.

(*****)

```

BIBLIOGRAPHY

1. Stanton, Tom. "The Kurzweil 4000 A State-of-the-Art Reader," PC, 4: 110-114 (July 9, 1985).
2. Bush, Larry F. The Design of an Optimal Alphanumeric Symbol Set for Cockpit Displays, MS Thesis AFIT/GE/EE/77-11. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1977 (ADA053-477).
3. O'Hair, Mark A. Whole Word Recognition Based on Low Frequency Fourier Complex and Amplitude Spectrums, MS Thesis AFIT/GE/ENG/84D-4. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1984.
4. Kabrisky, Matthew. Lectures for ENG 620, Pattern Recognition I. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, January 1985.
5. Routh, Richard Leroy. A Spoken English Recognition Expert System, MS Thesis AFIT/GCS/EE/83S-01. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, September 1983 (A136835).
6. Rich, Elaine. Artificial Intelligence. New York: McGraw-Hill, Inc., 1983.
7. Hayes-Roth, Frederick and others. Building Expert Systems. Reading, Massachusetts: Addison-Wesley Publishing Company, Inc., 1983.
8. Mandani, Dr. E. H. and Professor B. R. Gaines. Fuzzy Reasoning and Its Applications. New York: Academic Press, Inc., 1981.
9. Rauch, Herbert E. "Probability Concepts for an Expert System Used for Data Fusion," The AI Magazine, 5: 55-60 (Fall 1984).
10. Raviv, J. "Decision Making in Markov Chains Applied to the Problem of Pattern Recognition," IEEE Transactions on Information Theory, IT-3: 536-551 (October 1967).
11. Srihari, Sargur N. Tutorial Computer Text Recognition and Error Correction. Silver Spring, MD: IEEE Computer Society Press, 1985.
12. Shannon, C. E. "Prediction and Entropy of Printed English," Bell System Technical Journal, 30: 50-64 (January 1951).

13. Wesley, Leonard P. "Reasoning About Control: The Investigation of an Evidential Approach," Proceedings of the Eight International Joint Conference on Artificial Intelligence, 1: 203-206 (Aug 1983).
14. Garvey, Thomas D. and others. "An Inference Technique for Integrating Knowledge from Disparate Sources," Proceedings of the Seventh International Joint Conference on Artificial Intelligence: 319-325 (Aug 1981).
15. Zadeh, Lofti A. "Reviews of Books: A Mathematical Theory of Evidence," The AI Magazine, 5: 81-83 (Fall 1984).
16. Fagan, Lawrence and others. "Chapter V Understanding Spoken Language," The Handbook of Artificial Intelligence, Volume 1: 323-361, edited by Avron Barr and Edward A. Feigenbaum. Stanford, California: HeuristTech Press, 1981.
17. Levinson, S. E. and others. "Evaluation of a Word Recognition System Using Syntax Analysis," Bell System Technical Journal, 57 (5): 1619-1626 (May-June 1978).
18. Levinson, S. E. "The Effects of Syntactic Analysis on Word Recognition Accuracy," Bell System Technical Journal, 57 (5): 1627-1644 (May-June 1978).
19. Schurmann, J. "Reading Machines," Proceedings of the Sixth International Conference on Pattern Recognition: 1031-1034, Munich, Germany (1982).
20. Hagar, Hubert A. and E. Lillian Hutchinson. Words. New York: McGraw-Hill Book Company Inc., 1954.
21. Fries, Charles C. Linguistics and Reading. New York: Holt, Rinehart, and Winston Inc., 1965.
22. Ducera, Henry. "Computers in Language Analysis and in Lexicography," The American Heritage Dictionary of the English Language: xxxviii-xl, edited by William Morris. New York: American Heritage Publishing Company Inc., 1973.
23. Christie, Agatha. Sleeping Murder. New York: Bantam Books Inc., 1977.
24. Burrus, C. S. and T. W. Parks. DFT/FFT and Convolution Algorithms. New York: John Wiley & Sons, 1985.

VITA

Captain David Vincent Paciorkowski was born on 14 October 1956 in Worcester Massachussettes. He graduated from high school at Woodstock Academy in Woodstock, Connecticut in 1974. He attended the University of Rochester in New York for two years, studying in a premedical curriculum. After transferring to Worcester Polytechnic Institute in Massachussettes, he received the degree of Bachelor of Science in Electrical Engineering in May 1980. He entered the Air Force through the Officer Training School in April 1980 and received his commission in July 1980. After completing the Communications Officer Training Course at Keesler AFB, Mississippi, he was assigned to the 1815 Test and Evaluation Squadron at Wright-Patterson AFB, Ohio. There he served as an evaluation engineer, as a team chief, and as Section Chief for Standardization and Evaluation until entering the Air Force Institute of Technology in May 1984.

Permanent address: 7 Meadow Lane

Dudley, MA 01570

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED		1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE			
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/ENG/85D-31		5. MONITORING ORGANIZATION REPORT NUMBER(S)	
6a. NAME OF PERFORMING ORGANIZATION School of Engineering	6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION	
6c. ADDRESS (City, State and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433		7b. ADDRESS (City, State and ZIP Code)	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State and ZIP Code)		10. SOURCE OF FUNDING NOS.	
11. TITLE (Include Security Classification) See Box 19		PROGRAM ELEMENT NO.	TASK NO.
		WORK UNIT NO.	
12. PERSONAL AUTHOR(S) David V. Paciorkowski, BSEE, Capt, USAF			
13a. TYPE OF REPORT MS Thesis	13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) 1985 December	15. PAGE COUNT 139
16. SUPPLEMENTARY NOTATION			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB. GR.	
09	02	Optical Character Recognition, Contextual Postprocessing	
19. ABSTRACT (Continue on reverse if necessary and identify by block number)			
<p>Title: A CONTEXTUAL POSTPROCESSING EXPERT SYSTEM FOR ENGLISH SENTENCE READING MACHINES</p> <p>Thesis Advisor: Mathew Kabrisky, Ph.D. Professor of Electrical Engineering</p> <p>Approved for public release; UNCLASSIFIED LYNN E. WOLAVER 16JAN86 Dean for Research and Professional Development Air Force Institute of Technology (AFIT) Wright-Patterson AFB OH 45433</p>			
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS <input type="checkbox"/>		21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Mathew Kabrisky, Ph.D.		22b. TELEPHONE NUMBER (Include Area Code) 513-255-5276	22c. OFFICE SYMBOL AFIT/ENG

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

Knowledge-based programming techniques are used in an expert system to reduce uncertainty for optical character recognition by combining evidence from several diverse knowledge sources cooperating in an hierarchical configuration. The postprocessor development focused on a system for generic text input that is not constrained to a fixed vocabulary or particular subject domain. The key element to the system's effectiveness is a spell checking algorithm that is not operationally bounded by an enumerated lexicon or biased by statistical sampling. The postprocessor system is also design to interface almost any type of OCR front-end methodology.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

END

FILMED

3-86

DTIC